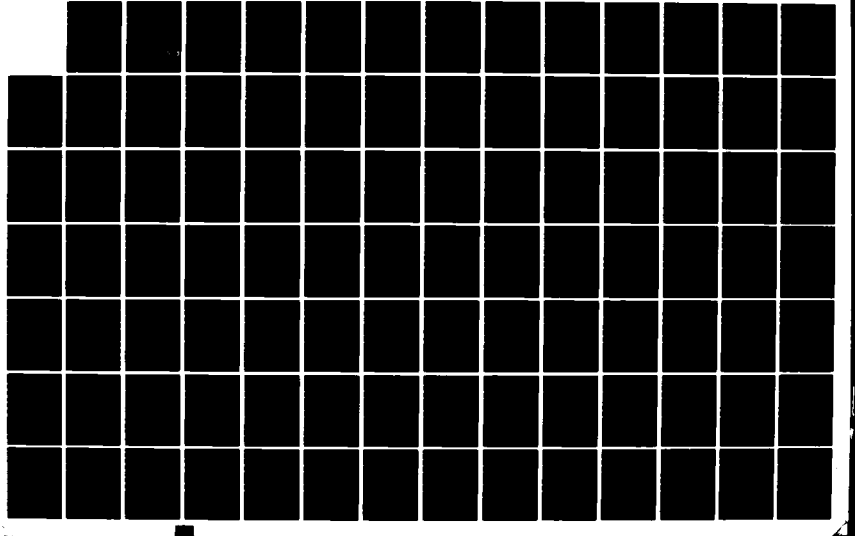
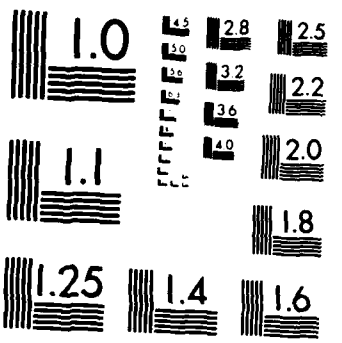
AD-A151 856 ANALYSIS AND SPECIFICATION OF A UNIVERSAL DATA MODEL FOR DISTRIBUTED DATA.. (U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. A J JONES

1/4

UNCLASSIFIED 14 DEC 84 AFIT/GCS/ENG/84D-11 F/G 9/2 NL





ANALYSIS AND SPECIFICATION OF A UNIVERSAL DATA MODEL FOR DISTRIBUTED DATA BASE SYSTEMS

THESIS

Anthony J. Jones
Second Lieutenant, USAF

AFIT/GCS/ENG/84D-11

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GCS/ENG/84

ANALYSIS AND SPECIFICATION OF A UNIVERSAL DATA MODEL FOR DISTRIBUTED DATA BASE SYSTEMS

THESIS

Anthony J. Jones
Second Lieutenant, USAF

AFIT/GCS/ENG/84D-11

Approved for public release; distribution unlimited

AFIT/GCS/ENG/84D-11

ANALYSIS AND SPECIFICATION OF A UNIVERSAL DATA MODEL FOR DISTRIBUTED DATA BASE SYSTEMS

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements for the Degree
of Master of Science in Computer Systems

Anthony J. Jones, B.A.
Second Lieutenant, USAF

March 1984

Preface

The purpose of this study was to analyze and define the system requirements for a universal data base, and then develop a universal data model to support a heterogeneous, distributed data base environment (the universal data base). The reason for attempting this development was the potential benefit to be gained from being able to network data base systems together.

I would like to acknowledge the great deal of support and encouragement that I received from my thesis advisor, Dr. Thomas Hartrum, and my reader, Dr. Henry Potoczny. I would also like to acknowledge the support I received from my friend 2Lt. Edward Jankus and my roomate John Pierce.

Anthony J. Jones

List of Figures

List of Tables

Abstract

A Universal Data Model (UDM) was developed for distributed Data Base Management Systems (DBMS). The primary goal was to allow for the effective communication between heterogeneous, distributed DBMSs. A system requirements analysis was first performed for a Universal Data Base (UDB). Three models were selected and investigated as candidates for the UDM: the Canonical, Entity-Relationship, and Relational. Due to the complexities of the UDB, the user was restricted to writing universal queries in a Universal Data Manipulation Language (UDML) and was restricted to only one version of each of the three prominent data models in use: IMS (heirarchical), DBTG (network), and System R (relational). Criteria were established and the relational model, augmented, chosen as the UDM. Data model mapping issues were examined and included discussions on distributed information, redundant data, the support of Third Normal Form, and target model specific issues. Algorithms were developed to show the mappings between the target models and the UDM. The Integration of these mappings were also addressed. The syntax of a universal data definition language and data manipulation language were described.

Table of Contents

ANALYSIS AND SPECIFICATION OF A UNIVERSAL DATA MODEL FOR DISTRIBUTED DATA BASE SYSTEMS

I. Introduction

Background

Since the inception of computers, the amount of data requiring storage and subsequent manipulation in average applications has grown rapidly. In the early 1960s, computers, on the average, worked with approximately 100K (K = 1024 or roughly 1000) characters. By the 1970s this amount had grown to 1 million characters and presently averages around 4 billion characters. It is predicted that at the present rate of growth the figure in 1988 will be approximately 10 million characters (8:19-20).

In the early fifties, as the amount of information being manipulated began to increase, the forerunners of the modern Data Base Management Systems (DBMS) came into operation. These were the early data definition facilities and report generators. While not DBMS by current standards they were the first step. The first real DBMS efforts, developed in the 1960s, were in-house efforts by individual companies who sought to improve their own computer operation. Each new DBMS developed within an organization maintained the same general characteristics of the previous DBMS while taking

advantage of improving data manipulation techniques and technology. The 1970s saw a shift from the in-house DBMS to proprietary development, with drives toward standardization and the mathematical and theoretical aspects of DBMS.

Data Base Management Systems. DBMSs are basically generalized data-processing techniques. A user stores information in a data base and uses the facilities provided by the DBMS to query, modify, and delete information in the data base as desired. A good example of a data base application is that of a university's student and class information. In a non-computerized system, information about what classes a student is taking or what classes a particular professor is teaching would be listed on paper and any requested information, a query, would have to be compiled by hand. In a non-DBMS computerized system, all of this information would be stored in the computer but in what are normally referred to as "flat files". Flat files are fixed width sequential files. A DBMS provides the user with a generalized capability to manipulate and structure data in a manner most convenient to him. Instead of writing special programs to evaluate any particular query, a user uses the DBMS's data manipulation language (DML) to interact with the information within the data base. A user may perform three basic types of interactions: queries, updates (including insertions), and deletions. A query involves "querying" the data base about the information in the data base. "What students are taking EE 5.87 Mini/

Microcomputer Lab at AFIT?" "Who is teaching EE 5.87 in the fall of 1984?" An update allows the user to modify data already within the data base to some new value or may involve adding a completely new value. A deletion involves deleting information contained within the data base. A DML replaces all of the specialized, unipurpose programs of the old information processing systems.

The data within the data base must not only be manipulated but also portrayed to the system and any application programmers. Data base systems provide data definition languages (DDL) to accomplish this. DDL's tell the system and any interested users how the information within the data base is organized and structured at the logical level. As an example, from a university's database, the DDL's description of the data base (schema) would indicate what information about a professor is being stored and how it is being organized. A possible analogy might be that of a reception desk and filing cabinet. Before secretaries (information/software systems), each user had to manually go to the file cabinet (computer) and get the information that they needed. With the advent of secretaries, a user could now go to the secretary and request information. Unfortunately, the first secretaries, as with most prototypes, only knew how to find one type of information and, therefore, many secretaries were needed. However, a new type of secretary (DBMS) came about which was given a very broad based, generalized type of

training. Therefore, given a small amount of instruction from the user, this secretary could accomplish all of tasks done by the previous other secretaries plus new, previously unspecified, tasks.

Advantages and Disadvantages. Data base management systems provide a great deal of power and flexibility. They offer such advantages as "elimination of program duplication, amortization of the one-time development costs over many applications of the program, and physical and logical data independence (8:22)." Unfortunately, all of these advantages are paid for in terms of operating efficiency and/or increases in resources such as hardware. The advantages, especially in light of today's decreasing hardware costs, outweigh the disadvantages. DBMSs are considered to be one of the most important and successful software developments in this decade and have had a significant impact on the field of data processing and information retrieval (6:3).

Distributed Environment. In recent times, DBMSs have increased in importance and usage while network technology and capability have also increased. These two factors have promoted the idea of logically and physically interconnecting independent, mostly heterogeneous, distributed data base management systems (DDBMS) together. The term "heterogeneous" means that the data base systems are using different approaches, called models, in representing and structuring the data stored in the data base. This is opposed to a

homogeneous environment where the different systems use the same model. These different models will be discussed shortly.

The goal of networking these DDBMSs together is to make these separate and otherwise independent and distinct data base systems appear as one large data base. In this context, the terms "independent" and "distinct" mean that the data bases don't share access to the same information in the same memory. They may, or may not, actually reside on the same computer. A majority of the time they will reside on separate computers/systems.

Given this data base network, the fact that the information desired by any particular user physically resides on several different data bases and, perhaps, in several different forms (i.e. models), would remain hidden from the user. In fact, where the information resides would be unimportant to the user in most circumstances. Initially, any such data base network would entail bringing currently established data bases together. However, this not only appears practical and advantageous for data retrieval and interaction, but also data base creation. Therefore, in the future, such a network would not merely allow a user to access information stored on other data base systems, but would also allow a user to create a new data base. These data bases could physically reside on one or more data base systems in whatever model (heirarchical, network, relational) most suited that any particu-

lar group of data to be contained within the new data base.

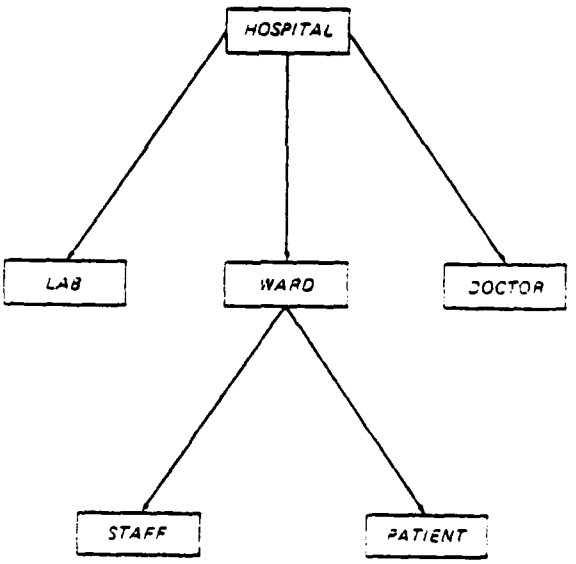
DBMS Models. Unfortunately, this goal presents several problems. The primary problem centers around describing and manipulating (i.e. modeling) the data stored in the data base. A second problem centers around each model's degrees of user control and responsibility. The models can be broken down into two catagories: procedural and nonprocedural. Procedural languages/models require that the user explicitly specify how the data base is to access the desired information. Nonprocedural languages/models require only that the user specify any predicates concerning what information should be retrieved, modified, or deleted. Of the three prominent models now in use, the first two, the heirarchical and network, are procedural while the last, the relational, is nonprocedural.

The heirarchical model (Figure 1), represented by a simple tree structure with each "node" in the tree having one parent (superior node) and any number of children (subordinate nodes), proves the most complicated and least popular of the three models. It was also the first DBMS and currently the most used. Although a natural way to model many real world environments and situations, the heirarchical model possesses certain anomalies for insertion, deletion, and updates. Additionally, the heirarchical model complicates its structures by distinguishing structurally between entities (records) and relationships between entities (expressed by

pointers). The heirarchical model's most serious drawback, the loss of symmetry (symmetric queries handled in non-symmetric ways), arises as a direct consequence of the fact that any given record only takes on its full significance when seen in the proper context (3:68). A record's full meaning is dependent on its position within the tree. Because of this,

> "the user is forced to devote time and effort to solving problems that are introduced by the hierarchical data structure and are not intrinsic to the question being asked (3:68)."

This all leads to unnecessary complications for the user. Finally the model described next, the network model (which is



HOSPITAL(Hospital code, Name, Address, Phone#, # of beds)
LAB(Lab#, Name, Address, Phone#)
WARD(Ward code, Name, # of beds)
STAFF(Employee#, Name, Duty, Shift, Salary)
PATIENT(Registration#, Bed#, Name, Address, Birthdate, Sex, SSN)
DOCTOR(Doctor#, Name, Specialty)

Figure 1. Hierarchical Model for Medical Data Base (9:149)

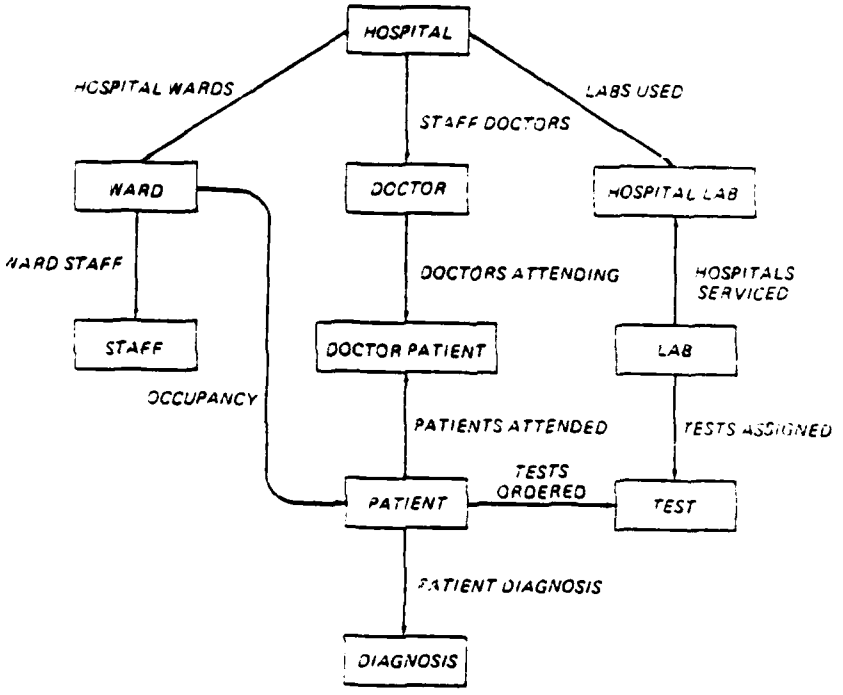
a generalization of the heirarchical model) provides all the capabilities of the heirarchical without its limitations and with more flexibility and ease of use.

The network model (Figure 2) allows a given record to have any number of parent or child nodes, thus permitting the user to model many-to-many relationships (i.e. many students taking many classes). "The network model allows the designer almost total control over the physical placement of the data (8:22-23)." The network model also maintains symmetry better. In addition, since a network model works with records and pointers, an experienced programmer can produce very efficient code and get maximum performance. Unfortunately, this requires a good programmer. Even with such a programer, an average application takes longer to design and code resulting in less productivity than with the next model, the relational. So while flexible and possessing potential for good performance, the network model's complicated nature proves a rather large drawback.

The relational model (Figure 3), considered the easiest to comprehend and use, organizes data into tables (called relations) made of rows (called tuples) and columns (called domains) with no implicit ordering among the rows of a table. Each column of the table contains atomic (indivisible) values with no repeating items within a column allowed.

> "The relational approach to data is based on the realization that files that obey certain constraints may be considered as mathematical relations, and hence that elementary relation theory

may be brought to bear on various practical problems of dealing with data in such files. . . . It is a characteristic . . . that all information in the database--both "entities" and 'relationships,'. . . is represented in a single uniform manner. . . .[a characteristic] not shared by the heirarchical and network approaches (3:65)."



HOSPITAL(Hospital code, Name, Address, Phone# , # of beds)
WARD(Ward code, Name, # of beds)
STAFF(Employee#, Name, Duty, Shift, Salary)
DOCTOR(Doctor#, Name, Specialty)
DOCTOR PATIENT(Doctor#, Registration#)
PATIENT(Registration#, Bed#, Name, Address, Birthdate, Sex, SSN)
DIAGNOSIS(Diagnosis code, Diagnosis type, Complications, Precautionary info)
HOSPITAL LAB(Hospital code, Lab#)
LAB(Lab#, Name, Address, Phone#)
TEST(Test code, Type, Date ordered, Time ordered, Specimen/order# , Status)

Figure 2. Network Model for Medical Data Base (9:121)

A concept arising from this mathematical basis is that of normalization. There are several different levels of normalization, each more stringent than the last. This discussion will limit itself to the first three normal forms.

First normal form (1NF) requires that every value in the relation, each attribute, is atomic. Second normal form (2NF) requires that every relation be in 1NF, and that "every nonkey attribute is fully dependent on the primary key (3:246)." Third normal form (3NF) requires that every relation be in 2NF, and that "every nonkey attribute is nontransitively dependent on the primary key (3:248)."

> "We choose to support only normalized relations in the relational approach because (a) . . . this choice imposes no real restriction on what can be represented, and (b) the resulting simplification in data structure leads to corresponding simplifications in numerous other areas--in particular, in the operators of the [data manipulation language] (3:86)."

Unfortunately, the relational model, while simple to understand and use, often turns out to be inefficient due to redundancy and the increased time it takes to perform the necessary operations (joins, selects, projections, etc. - see Chapter 4) to gain information from the different tables. However, the use of specialized data base computers will reduce these deficiencies (a current trend).

Summarizing, the heirarchical model, difficult to use, can be emulated by the network model. The network model, while very flexible and possessing potential for good performance suffers from a lack of ease of use and comprehension. The relational model, while easier to use and understand, suffers from performance problems.

The problem with these differing models not only revolves around their basic differences and tradeoffs, but also

centers on the fact that each model can prove best in certain situations. Thus, no best model exists at this time and

HOSPITAL

Hospital code	*Name*	*Address*	*Phone#*	*# of beds*
22	Doctors	45 Brunswick	923-5411	412
13	Central	333 Sherbourne	964-4264	502
45	Childrens	555 University	597-1500	845
18	General	101 College	595-3111	987

WARD

Hospital code	*Ward code*	*Name*	*# of beds*
22	1	Recovery	10
13	3	Intensive Care	21
22	6	Psychiatric	118
45	4	Cardiac	55
22	2	Maternity	34
13	6	Psychiatric	67
18	3	Intensive Care	10
45	1	Recovery	13
18	4	Cardiac	53
45	2	Maternity	24

STAFF

Hospital code	*Ward code*	*Employee#*	*Name*	*Duty*	*Shift*	*Salary*
22	6	1009	Holmes D.	Nurse	M	18500
13	6	3754	Delagi B.	Nurse	A	17400
22	6	8422	Bell G.	Orderly	M	12600
22	2	9901	Newport C.	Intern	M	17000
45	4	1280	Anderson R.	Intern	E	17000
22	1	6065	Ritchie G.	Nurse	E	20200
13	6	3106	Hughes J.	Orderly	A	13500
45	1	8526	Frank H.	Nurse	A	19400
18	4	6357	Karplus W.	Intern	M	18300
22	1	7379	Colony R.	Nurse	M	16300

DOCTOR

Hospital code	*Doctor#*	*Name*	*Specialty*
45	607	Ashby W.	Pediatrics
18	585	Miller G.	Gynecology
22	453	Glass D.	Pediatrics
13	435	Lee A.	Cardiology
45	522	Adams C.	Neurology
22	398	Best K.	Urology
18	982	Russ J.	Cardiology
22	386	Stone C.	Psychiatry

Figure 3. Relational Model for Medical Data Base (partial)

(9:96)

until an all encompassing model surfaces, these three models will continue in use. Furthermore , differently modeled systems cannot interact at this time easily, usually requiring a specially tailored system that depends heavily on the two models (rarely more than two attempted) in use and the data base management systems (DBMS) in question. This points out another complication. Even systems using the same model, unless using the exact same DBMS, may use different data manipulation languages (DML) and data definition language (DDL).

A possible solution to the problem of heterogeneous models involves finding a way of <u>universally</u> modeling, defining, and manipulating data stored in distinctly different models and systems. Through this universal model, universal definition language, and universal manipulation language, a

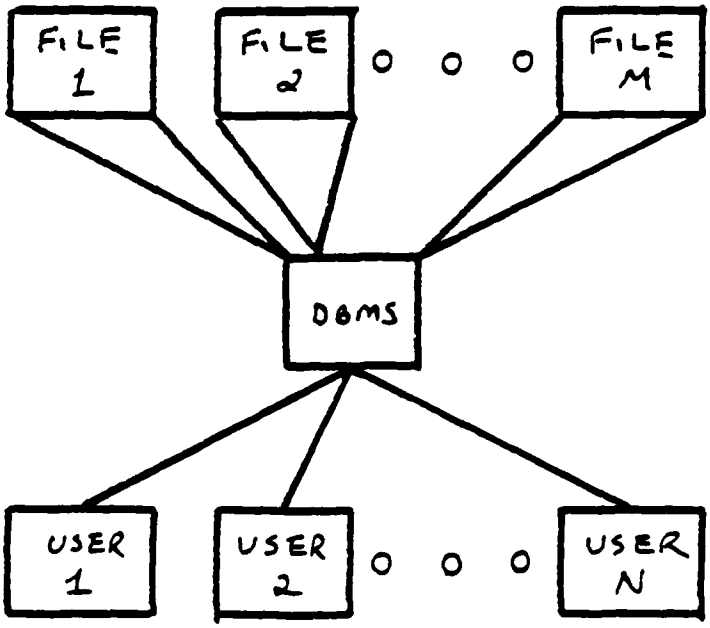


Figure 4. Integrated Views of Files.

data base, implemented in a model most efficient for that particular set of data, could still interact effectively with other data bases. In the same way that present independent data bases are "integrated", otherwise distinct files thought as one, now the objective is to integrate otherwise distinct data bases without changing their structure.

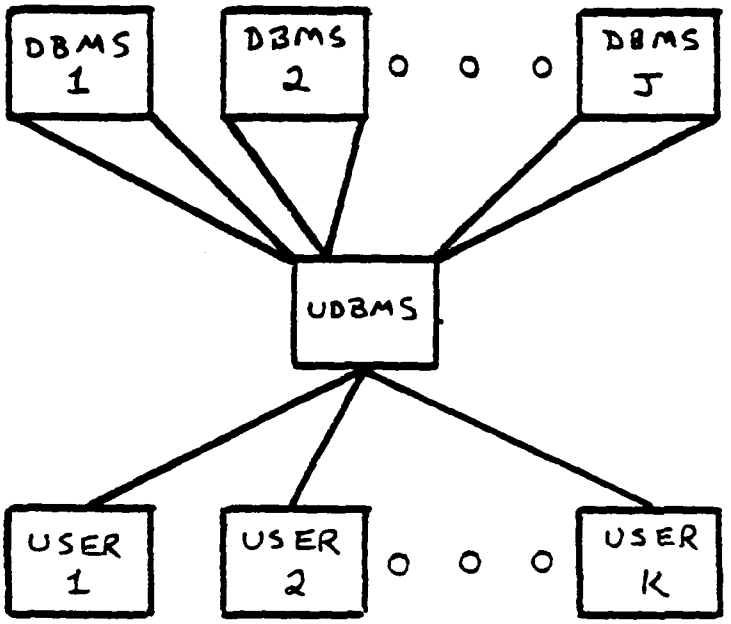


Figure 5. Integrated Views of DBMS.

Problem

Due to the existence and use of three different prominent data models and the many different DMLs, it is necessary to develop a Universal Data model (UDM), Universal Data Definition Language (UDDL) and a Universal Data Manipulation Language (UDML). The UDM, UDDL, and UDML will act as a central pivoting point for distributed data base management sys-

tems to effectively interact without forcing changes in each system's own model and DML. As a form of shorthand for this thesis, this problem will be referred to as the "UDB problem."

Scope

In this study the following aspects of a Universal Data Base system will be addressed:

1) Conduct in-depth research of current knowledge into the area of a Universal Data Base (UDB) to include an examination of the canonical, relational, and entity-based models to examine their usefulness and potential in solving the UDB problem.
2) Develop system requirements for a UDB.
3) Develop a UDM encompassing the three major data models in use today: the heirarchical, network, and relational.
4) Develop a UDDL encompassing the three major data models noted above.
5) Develop the mappings between the UDDL and the three major models noted above.
6) Discuss the various issues and policies involved in generating the universal representations of the individual local data base systems.

Assumptions

The assumptions for this thesis will center around the environment in which the UDM and UDML will operate. First, although the UDM will and should universally model all of the three aforementioned models, there exist variants of each of the models. This greatly complicates the issues and mappings. Therefore, for an initial attempt at the problem, this effort will assume particular variants of each model to work: the DBTG model for the network, the IMS model for the heirarchical, and System R for the relational model (the reader is directed to Date (3) for a description of each).

Secondly, the environment in which the UDB will work requires some basic description since in an open, unrestricted environment, it would prove very difficult for the UDB to function due to the complexities of the environment. Therefore, the following assumptions describe the environment in which the UDB will work (see Chapter 2 for rationale and an expanded list):

1) The environment will be such that a reasonable amount of standardization and cooperation can be secured. A military/government installation or a large corporate company are prime examples.
2) An organization, the Data Base Administration Center (DBAC), will perform the Data Base Administration functions.
3) Any new local data bases established after the UDB becomes operational will follow a specific and standard format established by the DBAC.
4) Any previously established data bases will remain as before, but every practical attempt will be made to convert them to the standard format.
5) The user will be required to comprehend the UDB and will use the UDML whenever information exists outside the local data base.

Summary of Current Knowledge

At present there exists a limited amount of research and information concerning a UDM, UDDL, and UDML. A fairly large amount of research has been done concerning subsets of this problem, such as translating from one model/system to another model/system and so forth. Recently, a great deal of attention has been focused on both the entity-based model developed by Chen (2) and the relational model as possible candidates for the universal model. Martin's canonical model (10) will also be examined for its possible use. Both Chen's and

Martin's models originally were developed to directly apply to the design process of a data base, but there exists growing evidence for and research into their possible applications in the area of a universal system. The relational model has long been advocated, and much debated, as the best model for most applications and, in recent efforts into this area, has been chosen as the global model. Direct research into the area of the UDB has been done by Date, Housel, Huang. However, most of these approaches take differing views and establish differing systems requirements for a UDB.

Approach

The first part of this study will involve an in-depth literature search of the relational, canonical, and entity-based models to determine their usefulness in this problem and their potential in general. Secondly, incorporating ideas from the literature research, recent research into the actual area of a UDB will be investigated. This will primarily center around Date's, Huang's, and Larson's work in the area, but will also encompass any other pertinent work uncovered. Following that, a set of system requirements will be established for a UDB. What defines a UDB? What must it accomplish? Next, a UDM will be developed to handle the three prominent data models. After that, a UDDL will be developed and the necessary mappings between it and the other three models will be examined. The UDML will also be developed but the mappings not addressed.

Overview of the Thesis

This thesis is an initial research effort by AFIT into the area of networking heterogeneous distributed data base systems together. This first chapter provides the reader with some background to data base systems in general and the UDB problem. The second chapter defines the initial systems requirements for the UDB. It will address the environment, the user, and the language. The third chapter discusses data base mappings in general. The fourth chapter describes the three models that are possible candidates (selected by this thesis) for the UDM. The fifth chapter compares and contrasts the three models in terms of the UDB and chooses one as the UDM. The sixth chapter will examine the DDL mappings and related issues in generating the UDB. The seventh chapter describes the syntaxes of the UDDL and UDML, and describes the role, function, and composition of the data dictionary. The eighth, and final, chapter summarizes the findings of the thesis, and discusses possible follow-on work.

II. Requirements:

The Environment, the User, and the Language

Introduction

Before designing a system or defining its requirements, one must consider the issues of the environment in which it will function and the type of users for whom it will provide service. A system for "casual" data base users will be significantly different from a system designed for experienced computer users. Similiarly, a system for an academic environment would differ radically from that of a system designed for a military environment. In a normal, homogeneous system, these issues would be important. In the more complex, heterogeneous environment, these issues are critical and will have a significant impact on the shaping of the UDB. A final concern considering the user and the environment, is what capabilities the UDDL and UDML must possess to perform correctly.

Overview

This chapter will first begin with a general discussion of the politics involved in shaping requirements and design decisions. The second section will discuss the environment in which the UDB can and will function. The third section will address the user and how he/she will view the system. The fourth section will describe the design objectives of the DML and the conclusion will summarize the main points of the

chapter.

Politics

"There is nothing more difficult to take in hand, more perilous to conduct or more uncertain in its success, than to take the lead in the introduction of a new order of things (11:1)." This quote by Machivelli establishes a good foundation for the problems that the UDB will face. Anyone who has ever dealt a great deal with organizations well realizes the problems and issues that can arise when attempting to introduce change in an existing system or method of doing something. This proves particularly true of large organizations, the prime target of the UDB, which possess the strongest defense against change - a bureaucracy. In general, people resist change. Employees will be reluctant to learn a new system. Management will be reluctant to invest time and money getting a new system and retraining their employees. This reluctance will be even greater when it involves cooperation with other organizations. This is not meant to unfairly project a negative view of large organizations, but rather to point out a well known fact that people, and hence organizations, resist change and sometimes are reluctant to cooperate, even when it is to their own benefit. This basic dilemma will and should influence the design and requirements analysis for the UDB. What types of organizations will want to use the UDB? Which organizations will be able to? Which organizations will not? In what environment can the UDB

function effectively?

The Environment

The functional environment of the UDB will be a complex one due to the political factors involved as well as the physical factors. Several observations about that environment follow. Each is described, expounded on, and conclusions drawn or deferred for later discussion.

> Observation 1: The current environment consists of independent, heterogeneous (or homogeneous) systems which have already established data bases, application programs, and procedures.

The heterogeneous nature of this environment applies not only to the data base management systems that exist but also the host computer systems. Further, heterogeneous, in the DBMS context, not only indicates differences in the model used but also the particular implementation of any given model (i. e. two DBMSs using the same model could be different because the model was implemented in a different way in each model).

> Observation 2: Given that the owners of these systems will be very reluctant, at least all at once, to replace their older systems with any new systems, rewrite their application programs and/or retrain their people, the UDB will have to function in such a way as to minimize any of the aforementioned activities as much as possible.

Obviously, from a requirements standpoint, it would be optimal to have the situation where the user could simply be informed that his system has been expanded to include more

information and other than that, nothing has changed. From the design view, it would be very desirable to make everyone switch over completely to a single new system. Neither of the above viewpoints prove to be practical ones. The design point of view is impractical for already mentioned political and cost reasons; and the requirements point of view because it presents some very tough problems. The first of these problems is that if the user is to view all the data as in his local system, then how will the data actually outside his system be presented to him? In what form will this data be presented? His local model may not be able to express the structures and constraints of another model effectively. Should a new, unrelated model be used to present all of the global data to the user? If so, which model should be chosen? This will, of course, force the user to learn the new model. If this model proves so flexible, why not use it instead of the local one altogether? Furthermore, what language will the local users work with? Will they work in their local language, which would be translated into the universal language, or will they have to work in the universal language? These issues will be discussed later in this thesis and a conclusion or approach decided upon. Finally, it should be noted that the independent local systems might actually be on the same machine but are considered distinct data base systems.

Observation 3: The UDB will function in an environment in which the users are reasonably cooperative and non-threatening to each other.

One of the attractive properties of a universal system, besides merely increasing the amount of available information, is that if certain subsets of information, e. g. personnel information, seemed particularly suited to a particular model, e. g. the relational, then all of the personnel information of the independent systems could be moved to a different data base system on the network. This is certainly one of the more extreme benefits, or possible uses for the UDB, but it does illustrate its potential. However, this type of utilization, as well as any less elaborate use, requires a cooperative and non-threatening environment. Even simply allowing users access to other user's information requires cooperation and a "non-threatening" environment. Data security and integrity present problems. Users, perhaps competitors, could seek to illegally access or modify information of other users. At present, data base security methods do not provide cost and performance effective protection.

Observation 4: The UDB will best function within a large governmental, military, or civilian organization.

Unfortunately, at this time, it proves rather impractical to have the UDB tying together all of the data bases in the "world". However, this does not really detract from the value of the UDB. For the most part, it is large organiza-

tions that have the most need for the UDB and the greatest otential gain. Eventually, perhaps a nationwide or world-wide UDB setup might be possible but not at this time. The complex issues of data security prove too difficult and unreliable.

> Observation 5: Standards and policies established for the UDB must be enforceable and enforced.

This observation is an obvious one. Standards and policies will increase the effectiveness of almost any system. This will prove even more beneficial, and in fact necessary, in the environment of the UDB.

> Observation 6: Similiar to a data base administrator, some sort of "universal administrator" will be required to create and enforce standards, and perform other DBA type functions. This organization will be called the Universal Data Base Administration Center (UDBAC).

Clearly, as a local DBMS requires a DBA, the UDB will require some similar type of entity and the responsibilities of the UDBAC will entail a great deal more work and cooperation. The UDBAC will have to coordinate all of the activities of the data bases on the system. This will certainly require more than one individual, hence its name.

> Observation 7: The prime objective of the UDB is to develop a model to allow the prominent three models to interact.

The primary objective is not to develop a new superior

model. The development of a new superior model, while certainly beneficial, would not necessarily solve the problem of networking the existing data base systems together. As noted earlier, organizations are not going to want to give up their present systems and investment, even for some new superior model. If the universal model proves superior then the various organizations may eventually convert. However, this will hopefully not be a requirement for becoming part of a UDB system.

> Observation 8: There are particular instances where each one of the three present models proves to be the best model for that instance. Their elimination from use may, therefore, not be optimal unless the new model proves able to recognize all of the strengths of the three models without their limitations.

Assuming that the new universal model does not prove to be a superior model, having the different models will prove useful to the users of the UDB. It is certainly true that there exist data base instances wherein each particular model will prove to be the best for that instance. Therefore, a user of the UDB could have, for example, several relational and network data bases, and a few heirarchical. When a user decides to establish a data base, the user or UDBAC can choose which one of the three models it would best fit. The user, who may be used to working on relational systems, will, in some fashion, deal with the new data base in either the relational model or the universal model.

Observation 9: The UDM, UDDL, and UDML will all be logical in nature but also designed to actually appear as a normal local DBMS.

The purpose of this restriction is to present the UDML, UDDL, and UDM in a manner most familiar to data base users. The logical model, while truly only logical in nature, will come across as an actual DBMS. It should be noted that it may prove necessary to provide the UDBMS with actual relational operator power (See Chapter 6).

Observation 10: The UDB will support all of the functions that a regular DBMS would support, i.e. retrieval, updates, deletion, etc. However, the UDBAC will formulate policy on deletion and updates to ensure the proper operation of the data base and protect the rights of the local systems.

Obviously, allowing updates and deletions on a universal basis presents some dangers and problems. However, the same policies that now govern local DBMS updates and deletions should prove, for the most part, applicable to this situation, if stringently enforced. Most users will only require and be allowed the capability to query the universal system. The UDBAC will formulate the necessary policies to allow for someone, other than the local system users, to modify information stored on the local system. Furthermore, in the same vein, security levels or views might be used to limit unauthorized access and modification. This can be handled at both the local and universal level (through the universal data dictionary). As noted earlier, any security provided

will not be foolproof.

> Observation 11: Each local system will have some sort of local data directory/dictionary. Furthermore, a global data directory/dictionary will also exist in some fashion.

The local data dictionary will be in the local model. The global data dictionary will be in a global/universal model. However, where will the global data dictionary reside? Will it reside at one local DBMS and be able to be passed around, if necessary? Or will it reside at some computer system solely dedicated to that function? Will the local systems have a portion of the global data dictionary or will the local systems have to query to master copy of the global data dictionary (see Observation 16)? The global and local data dictionaries will prove to be an extremely important entity in the UDB. Chapter 7 discusses the data dictionary in more detail.

> Observation 12: The UDB will function in a response time which is not significantly different than that of the local system(s).

Once again, this may appear to be a statement of the obvious but it is a relevant observation and an important one. The UDB will not be an off-line process but rather will respond to queries as if the information were all on the local system, insofar as this is possible.

> Observation 13: A physical network will exist to allow the data base systems to communicate.

Observation 14: It is uncertain at this time what percentage of queries will involve non-local information.

One can conjecture, with reasonable chance of success, that initially the large majority of the queries will remain local. However, as the users get accustomed to the UDB and realize its full potential, the percentage of non-local queries will certainly increase. With this increase in use, the traffic on the network will increase correspondingly.

Observation 15: In a network environment, communication costs are the prime consideration in terms of cost. Therefore, the UDB should attempt to minimize communication not directly related to queries.

Although the scope of this thesis does not directly involve network considerations or cost considerations, it is important to realize that these are still environmental factors which should be taken into consideration. This thesis is focusing on the the model and DDL mappings, but is also examining the UDB problem and system as a whole. Therefore, while some of these observations don't directly pertain to this thesis effort, they are presented for completeness and for the benefit of any follow-on work.

Observation 16: By virtue of observation 11 and 15, the global data dictionary will reside on some separate system as a master copy. Each local system will contain some portion of the global data dictionary for its own use, querying the master global data dictionary only when necessary.

By having some portion of the global data dictionary, this will reduce the amount of network interaction and response time as each query will no longer always have to ask the global data dictionary (which would require a network communication) for information. An extreme version of this would involve having each local system possess its own copy of the global data dictionary. This would eliminate all queries from the local systems to the global data dictionary on the network but would also make maintaining the global data dictionary more complex.

Observation 17: Via observation 15, all transmissions of information from global DBMS should be all-records-at-a-time (6:7).

Observation 18: The type of transaction that will occur over the UDB will be : 1) queries, 2) data transfer, 3) global data dictionary updates, 4) UDBAC messages, 5) updates, and 6) deletions.

Observation 19: The UDB will function in an environment which will dynamic in the way it changes.

Imagine the number of minor structural changes (adding/deleting a relation, relationship, record, data base and so forth) that occur at an LDBMS on any given day. Now imagine the number that could occur in a UDB system. Obviously, the UDBAC will have to formulate policies on this subject. It is highly probable that the entire contents of a LDBMS will not be a part of the UDB or available to all users. Furthermore, the UDBAC will, in all likelihood, have to regulate such

changes.

With the environment in mind, it is now time to examine the user's view of the system and possible approaches that can be taken to accommodate that view.

The User's View

The important issue in terms of the user's view of the universal system centers on how the contents and structures of the universal system are presented to a user of any given local system. Should the data in the universal data base be presented in terms of the local model or the universal model. Could some combination of the two work? This same debate centers around which language (local or universal) the users will be allowed to use. Each possibility has advantages and disadvantages.

The local model approach (Figure 6) is appealing from the user's and requirements viewpoint. With this approach, the only apparent change to the user is the enlargement of the existing data base. This will require a minimal amount of effort on the user's part to incorporate the UDB into his/her system. Little or no retraining will be required. Unfortunately, this places more of a burden on the system. First, a two level translation must now take place (source LDBMS --> UDBMS --> target LDBMS). Secondly, and most importantly, how will the data within the universal data base be presented in the local model? Will every local model be able to accurately depict all of the information? Will a

given query expressed globally result in the same set of information returned as that same query expressed locally (known as query equivalency). Assuming that these questions are sufficiently handled, then every local system will require its own unique representation of the universal data base. Furthermore, every time a modification to a data base occurs, a new set of representations must be generated for every different system.

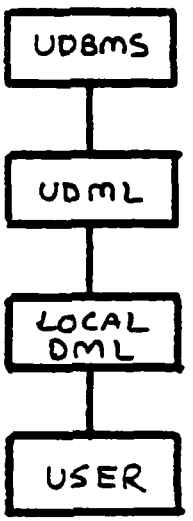


Figure 6. Local Model Approach

The universal model approach (Figure 7) is appealing from the system's and design point of view. If all queries are done in the universal model, then only a one level translation is required (UDBMS --> target LDBMS). Also, only one representation of the data will be necessary and any changes will only require the generation of one new representation. This approach does not imply that the local model is replaced but rather that queries are done in the UDML. Local DML

queries could still be possible if the queries involved only local data.

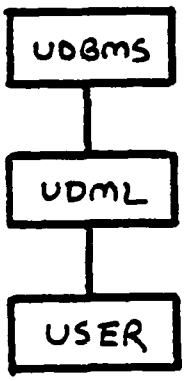


Figure 7. Universal Approach

A combination approach (Figure 8), or bi-model approach, has all of the disadvantages of both systems and virtually none of the advantages. In this approach two models are used. One model, a logical one, is used to convey the contents of the data base to the various users. The second

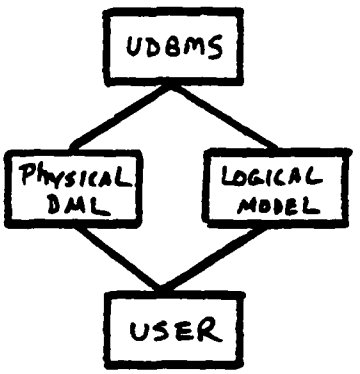


Figure 8. Bi-model Approach

model is the model that is actually used to process queries, etc (1). This approach requires the users to know and under-

stand two new models. For this reason, this approach is not considered a viable alternative.

A fourth approach (Figure 9) is a compromise of the first and second. This approach involves using one of the three local models but only one variation of each. For example, the UDB would allow a network user to work in the network model, but it would be a standardized network model, say DBTG, for all network users on the system. The same would go for the relational and heirarchical users. This approach has the advantage of not forcing the user to learn a completely new model. The user will still have to learn a new system, and rewrite application programs, etc., but it will at least be a similiar model to his old one.

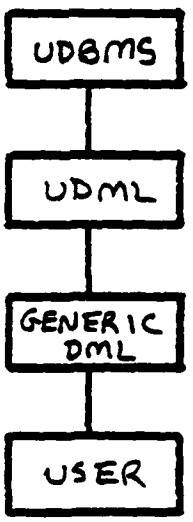


Figure 9. Generic Approach

A fifth approach (Figure 10) is a variation of the fourth. Once again a generic representitive of each of the three models is chosen. However, the users still work in the

local model/language, but the local transactions (queries, etc.) are then mapped into the appropriate generic version. The generic transaction is then mapped into the UDML. This offers several advantages over all of the previous approaches. The users are allowed to work in their local models, but the UDB still avoids having to map a large number of different models (and variations) because it only has to map between the generic versions of the models. The trade-off is the fact that now the UDB has an additional mapping to go through for each global transaction. However, the additional mapping to the generic versions is comparatively a trivial mapping.

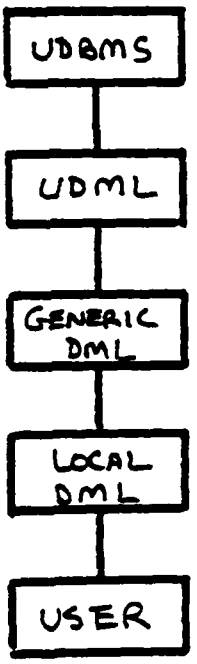


Figure 10. Generic-Local Approach

In comparing the five approaches described above, the last approach certainly has the most advantageous properties.

Unfortunately, one must now consider how the users are going to view the data base. In the fifth approach they would view it in their local format. However, as pointed out earlier, this could prove very difficult and impractical to accomplish. This problem eliminates the first and fifth approaches (along with the third already eliminated). The second and fourth approaches both require the user to learn a new model. The fourth has the advantage that the user will be working with a model similiar to his original and the format will also be similiar. Unfortunately, as with the first and fifth, the local format proves too difficult to support. Therefore, the second approach is deemed the best approach.

Now that the general requirements of the UDB have been analyzed, it is time to look at the general design objectives which should be used in developing the UDML.

The Language

The following design objectives indicate the desired traits for the UDBMS (UDDL and UDML). It is acknowledged that some or many of them may not be practical or possible depending on the model chosen for the UDM. Obviously, they should impact on that choice but they will not be the overriding factor(s).

> Design Objective 1: The UDBMS should be a user friendly high-order language which supports all three of the prominent data models now in use.

Designing the UDBMS to be user friendly and of a high-order language nature will make the UDBMS easier to use and understand. Supporting the three models is an obvious objective. By supporting, it is meant that the UDBMS will be able to map constraints, structures, and operations from the UDM to the other models and vice versa (see Chapter 6).

> Design Objective 2: The UDBMS should support the three models with a single, integrated set of facilities, not three separate ones (4:190).

Several benefits can be derived from this objective. Primarily, the benefit applies to a user only having to know one language and UDBAC (and local DBAs) only having to maintain one language. Additionally, if all users of the system know the UDBMS, there exists a common medium for them to interact.

> Design Objective 3: The UDBMS should stress the user's view over the system's view.

Basically, this means the the UDBMS, and UDB in general, will attempt to minimize the amount of knowledge of the underlying schemas and structures that a user must know. Any given user will only have to view the UDB in one model. This one model will either be the universal model or the local model (see Chapter 5).

> Design Objective 4: The UDBMS should be independent of any particular computer system or hardware (4:190).

Design Objective 5: The UDBMS should support both procedural and nonprocedural operations (7:84).

The UDBMS will have to be designed to handle the difficulty of nonprocedural source commands being processed against a procedural target data base (no path specified) and a procedural command against a nonprocedural target data base (path specified but no real path exists). Some of the problems posed by this problem can be handled in the mapping process. The UDB could determine the correct path if none was specified but, of course, ambiguous situations could still arise. The UDBMS itself could be designed to ease or eliminate some of these possible problems. It is important to note that it may not be practical nor desirable to actually support procedural operations. It may prove too complex to navigate through a distributed, heterogeneous data base.

Design Objective 6: The UDML should provide a full range of navigational operations at the record level (4:190).

It is unclear whether or not it will be possible, and if possible, practical to support navigational operations in a distributed, heterogeneous environment. Besides the increased communication required as the user navigates through the distributed UDB, there is also the problem of how to navigate through a relational system, which may in turn be an intermediate position to another procedural system.

Design Objective 7: The UDML should provide a full range of derivational operations at the set level (4:190).

Design Objective 8: The UDML should be able to perform retrieval, update, and deletion operations.

Design Objective 9: The UDBMS should be able to handle one-to-one, one-to-many, and many-to-many relationships.

It should be noted that while a model supports, for example, many-to-many relationships, it might not support a theoretically correct many-to-many relationship but rather a pseudo one. The UDBMS will seek to express all three of the above relationship types in a theoretically correct manner.

Design Objective 10: The UDML should provide embedded and interactive capabilities which do not significantly differ in syntax or operation.

By standardizing the snytax, it will prove easier to use the UDML's different versions (embedded and interactive).

Design Objective 11: The UDML should provide direct reference capability in the embedded and interactive version (4:191).

Direct Reference is treating the data within the data base as if it were a regular part of the program or local work space.

"The basic point is simply that data in a database is in the system . . . it should not be necessary to move it from one place to another in order to process it - it should be possible to access it directly, just as it is with ordinary . . . or 'local' data. A comparative uniformity of reference for local and global data is a great simplifying factor for the user (4:191)."

Design Objective 12: The UDBMS should permit null values.

The primary reason in supporting null values is that some of the LDBMS systems support null values and it would be easier to present them and map to and from them if the UDBMS supported null values also.

Design Objective 13: The UDML should be designed to be as easily parsed as possible.

This quality is necessary due to the distributed nature of the UDB. A query may come into the system in which various portions of the original query must be sent to different LDBMS. A highly parseable UDML will aide greatly in this process of distributing the query.

Design Objective 14: The UDML should have separate constructs for selection and action specification (5:100).

This will increase the parseability and the modularity of the UDML. It provides a more flexible and powerful interface. Data selected in one block may be required in the selection block of another nested block, but may never be output to the user (5:100).

Design Objective 15: The UDML should support selection nesting.

Design Objective 16: The UDBMS should explicitly state all constraints.

This will make the UDBMS more flexible and mappable since the UDBMS should assume that there are no intrinsic constraints within the structure of the UDM. See Chapter 3,

Data Model Mapping, for a discussion of constraints.

> Design Objective 17: The UDML should support both record-at-a-time and set-at-a-time capability.

This will only be an objective if procedural operations are supported. If not supported, then only set-at-a-time operations will supported.

> Design Objective 18: The UDML should be able to hold any number of positions within the data base and these positions will be by explicit program command, if applicable (4:190).

Objective 18 will also only be supported if procedural operations are supported.

> Design Objective 19: The UDDL should define relationships in a flexible and powerful manner. The primary way that this could be achieved would be through not forcing the user to physically predefine relationships.

This is an important objective because permitting the user not to have predefine relationships improves the power and ease of use of the system. Heirarchical and network based DBMS are examples of systems/languages which require the physical predefinition of all relationships (by pointers). Any additions or deletions of relationships could cause a significant amount of schema restructuring. A relationally based DBMS is an example of a system/language in which relationships are not predefined. The desired information is placed in various tables and how these tables are related is, to a great extent, dependent on how the user's

DML commands are written. Essentially, the user defines the required relationships as he/she needs them.

Conclusion

In this chapter, the basic environmental, user, and language requirements have been examined. In many cases whether or not a particular objective is supported is very dependent upon the language chosen as the UDM and how the UDB evolves over the remainder of this thesis. After these decisions are finalized, the exact objectives of the UDB will be stabilized.

III. Approaches to Supporting a Multi-model System

Introduction

The next step in developing a UDB involves determining how to support, in a general sense, a multi-model system. Somehow all the models in the system must be represented and incorporated into the UDB. Two basic approaches exist that accomplish this goal, the mapping approach and the composite schema approach (7). The mapping approach suggests a direct or indirect mapping of one schema to another. The composite schema approach develops a common schema by embedding the schema for one data model into a schema of another model which may in turn be embedded in some other schema of some other model.

Overview

The approach to be taken in this thesis will be a type of mapping approach which will involve an intermediate data model. However, in the interest of comparison and contrast, the composite approach will be briefly described. The mapping approach will be discussed in detail and will focus on the different types of mappings and various considerations involved. The conclusion of this chapter will compare and contrast the two and discuss why the mapping approach was chosen. The discussion of the mapping approach in this chapter is almost exclusively a paraphrasing of the 14th chapter in

Tsichritzis and Lochovsky's book on Data Models. Citations are, therefore, excluded. It is highly recommended as a reference in the area of data models.

The Composite Approach

The composite approach embeds a schema for one data model into the schema of another model. A single schema supports the DML commands of all the models in the common schema. This process can, of course be recursive. The common schema produced contains objects which can be viewed as an object of any of the models. This property allows the user to take advantage of the strengths of the various models. Thus, if a situation required a heirarchical model, then that subset could be used. The same could be said for the other models. This method is often referred to as an "onion-layered" approach (7:91).

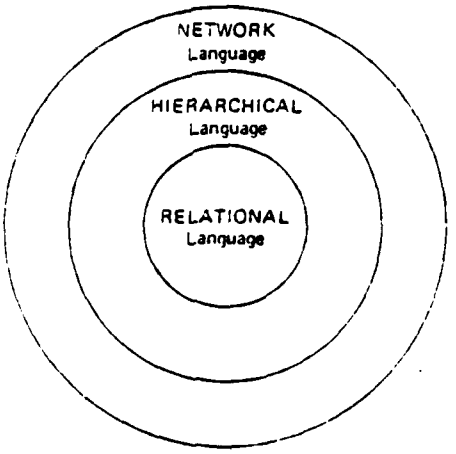


Figure 11. Onion-Layered Approach (3:451)

The Mapping Approach

> "The Mapping approach takes a source schema and maps it directly or indirectly into some specified target schema of another model. Data manipulation commands generated from the target schema are then translated to an equivalent command which can be processed against the source schema (7:84)."

In mapping between data models, four aspects must be considered in the process: structures, constraints, operations, and data bases. Since the universal system, towards which this effort is directed, requires that the two schemas be equivalent, the aspects of structures and constraints must be considered as a single entity. This entity is also known as a data model.

> "Two schemas are equivalent if (1) they describe the same data base, (2) commands expressed in terms of one schema can be translated into commands in terms of the objects of other schemas, and (3) the same changes are made to the data base if any source or target commands are executed (7:84)."

Further, the mapping process of a data base, where one data base is permanently mapped into another, has no relevancy to this thesis effort and hence, will not be addressed. This is due to the fact that the UDB is to function as a communication focal point. An actual transformation from one data base to another is not necessary nor desired. Therefore, this leaves the mapping aspects of data models and operations. Both of these aspects can be considered separately and could apply to either a heterogeneous or homogeneous environment. Finally, the resulting mappings can be described as constructive or nonconstructive.

> "A constructive mapping is one in which a data base (instance) according to one schema is mapped to another data base (instance) according to another schema. In nonconstructive mapping, the target data base may exist, but it is not obtained through the mapping process (9:301)."

Within this context, there are eight distinct types of mappings. These different mappings are generally distinguished by their constructivity, heterogeneous/homogeneous nature, and whether or not operation mapping occurs.

Table I. Data Mapping Types

Nonconstructive Mappings
Schema Restructuring
View Mapping
Schema Translation
Operation Transform
Constructive Mappings
Data Base Reorganization
Homogeneous distributed System
Data Base Translation
Structure and Constraint

Nonconstructive Mapping. Nonconstructive mappings fall under one of four types: schema restructuring mapping, view mapping, schema translation mapping, and operation transform mapping. The first two occur in a homogeneous environment while the last two occur in a heterogenous environment. Schema restructuring mapping involves schemas based on the same model in which operations are not mapped. In this mapping, if the source schema data base exists, the resulting target schema data base will be virtual and is never actually constructed. View mapping is similiar to schema restructur-

ing except that operations of the target schema are mapped to the source schema's data base. This situation often arises from a desire to provide different subschemas of a schema to different users. Schema translation mapping is the same as schema restructuring except that it is being done in a heterogeneous environment. This arises during schema design when it is desireable to express the requirements of an application according to one data model and then implement it according to another. The last nonconstructive mapping, operation transform mapping, is similiar to the view mapping except it also is in a heterogeneous environment.

Constructive Mappings. Constructive mappings also fall under one of four types: data base reorganization mapping, homogeneous distributed system mapping, data base translation mapping, and structure and constraint mapping. The first two types of mappings occur in homogeneous environments while the last two occur in heterogeneous environments. Data base reorganization mapping involves no operation mapping and requires an algorithm to obtain the target schema data base from the source schema data base. This mapping type incorporates schema restructuring since the target schema must somehow be mapped from the source schema. When operations are included in the mapping process, the process is called homogeneous distributed system mapping. This situation results from a system with several different schemas and associated data bases all using the same data model. This mapping re-

quires a global schema that can encompass all of the different schemas. Operations are then mapped against this global schema to appropriate operations on different underlying schemas. The data base translation mapping is the same as the data base reorganization mapping except that operations are mapped and it is in a heterogeneous environment. The data base cooperation mapping is the same as the homogeneous distributed systems mapping except that it is in a heterogeneous environment.

In the UDB environment the resulting mapping will be used to formulate the correct instructions for the appropriate LDBMSs to process. Thus, the required mapping is nonconstructive. Since operations must also be mapped, the UDB will require an operation transform mapping. For this type of mapping, two things must be mapped, the data model and the operations.

Data Model Mapping

As stated, data model mapping is the consideration of structures mapping and constraint mapping together. The basic structures of data models are derived from the concepts of sets and relations. This fact makes the mapping of structures relatively easy since most data models are defined in these terms. The real difficulties arise in attempting to map constraints between models.

"A constraint can be thought of as a property of the schema which should be true. It can also be thought of as a

natural relationship between some attributes or entity types (9:275)." Constraints can also be considered as defining restrictions on the domains within a data base. Constraints are expected to be true for any and all structures within the data base (schema) for which a given constraint applies. An example constraint would be the fact that any given employee has only one social security number and/or that any given employee does not make more money than his/her manager. These constraints disallow an employee having two or more social security numbers or an employee making more money than his/her manager (9:11). Constraints are expressed either explicitly, implicitly, or both. Explicitly stated constraints are considered better because the user has total control over the constraints placed on the data and is not as limited as would be the case of implicit constraints. An example of an implicit constraint is shown in an IMS (heirarchical) structure. In the IMS medical data base example (see Appendix F), information about a doctor cannot be placed within the data base until that doctor works for a hospital. This is not explicitly stated within the schema, but rather is a direct result of the heirarchical structure. With explicitly stated constraints, the user or programmer will not be hampered dealing with unnatural data constraints.

The difficulties in mapping constraints center on the fact that some constraints are almost completely inherent within the structures of some models, partially so in others, and

not at all for some. This proves true even when mapping between DBMSs based on the same model since, in actual implementation, systems using the same model may still differ slightly. One possible solution to this schema translation problem involves developing a mapping algorithm between a specific pair of data models. Most often this is termed a derivation of mappings among the relational, network, and/or heirarchical schemas. Another approach is to use a mapping data model as an intermediary representation between any pair of data models. In this approach, it is necessary to map only between the intermediary model and any other data model, requiring at most 2N one-way mappings in a system with N schemas, rather than between every pair of data models, requiring N(N-1) mappings (assuming no duplications). At present there exists very little formalism in this area of schema translation mapping. Two issues in this are unique keys for relations and representing links and set membership options in a relational schema. Every relation requires a unique key. However, not every record type in a network requires a key at all. Somehow a unique key must be constructed for each record type in a network (or heirarchical) schema. One approach is to use a data base controlled attribute which, unfortunately, is visable to the user as a system controlled attribute. Another approach is to construct a unique key for each record type from specified record keys by propagating data items along links. The second

problem of representing links and set membership requires the specification of explicit constraints in, for example, a relational schema. This may require additional relations and the presence of null values in the relational data base.

Operation Mapping

In the heterogenous environment, operation mapping requires an operation transform type of mapping. The approaches to this are the same as stated for data model mapping: direct operation mapping or mapping through an intermediate model. Unfortunately, very little formalism exists in this area either. However, an outline of some of the issues may prove of value.

The first issue is query equivalence. Query equivalence "is the determination of whether a query specified on one schema will give the desired result when executed on another schema (9:320-321)." Problems arise between different models when, for example, one is mapping a schema to another schema which must be able to identify the appropriate paths to take in acquiring the necessary information. This type of mapping is particularly difficult when conjunctive (OR) terms appear in the selection criteria.

Problems also arise when mapping between navigation type operations and specification type operations (i. e. operations which specify criteria for the desired data). Specification to navigation operations are relatively straightforward since the mapping is nonprocedural to procedural.

Navigation operations to specification operations (or even navigation to navigation) prove much more difficult since navigation operations use "currency pointers" extensively. Currency pointers indicate where in the structure of the data base the system's attention is focused. Capturing currency indicators in a specification language is difficult. This is often compared to decompilation in programming languages. One possible method is to infer a general pattern for a specfication statement by grouping and analyzing the navigation statements. Another mapping problem involves updates. The problem centers on the fact that different data models have different side effects on modifications because of the presence of implicit and explicit constraints. An example would be a deletion of some record in a heirarchical schema. How should the users of other systems view this? If other users are to view it, how should they view it?

Conclusion

Two approaches have been presented in this chapter and both have their advantages and disadvantages. However, the important factor in deciding between the two centers on the requirements of the UDB. The composite approach embeds schemas one within another, but can this approach be generalized efficiently, in a stand alone language, for an arbitrary number of DBMS with N distinct schemas? It would seem that the resulting schema, assuming that it could be done, would be extremely complicated. For example, there are many differ-

ent actual implementations of a network model. This proves to be true for the other two models also.

> "The primary difference between the two, from the user's perspective, is that the composite approach allows the mixing of commands from the different schema's commands. The mapping approach restricts the user to one type of command (7:86)."

Although this statement may be correct in general, in the UDB's case the goal is not to allow a user to mix different model's commands but rather to allow the user to communicate with other models using his/her own model's commands, or perhaps the universal language's commands. Therefore, while an advantageous property, it has no bearing on the UDB application. Clearly, the mapping approach provides more flexibility in a heterogeneous environment.

IV. Universal Model Candidates

Introduction

In Chapter Three the conclusion was drawn that the correct approach to providing a universal interface between the three prominent data models was to use an intermediate mapping model. This intermediate, logical model is to be the focal point of interaction between the different DBMS of the DDBMS network. Any query that involves information that resides on more than one DBMS of the DDBMS network will be mapped into the "universal" model and then parsed and mapped back into the appropriate local model(s) to be processed by the corresponding LDBMS. The decision that must be made, therefore, is what model is to be this "universal" model?

One possible approach to choosing this "universal" model would be to use one of the three prominent models now in use. A second approach would be to choose a "logical" model as the UDM. For this thesis three different models were chosen for investigation: the canonical model (Martin), the entity-relationship model (Chen), and the relational model (Codd). These three models represent the range of models now in existence. The canonical model, representing one extreme, is a purely logical model with no connection to "reality". The entity-relationship model represents the middle ground. It is a logical model but bears a strong resemblance to the models now in use. The Relational model represents the opposite end of the spectrum from the canonical.

It is a model actually in use and, of course, is one of the three models that the UDB must work with.

Overview

The purpose of this chapter is to present the possible candidates for the UDM that this thesis effort has decided to evaluate for their potential in solving the UDB problem. Each model will be briefly described to give the unfamiliar reader some idea of what each of the models is like.

The Canonical Model

The Canonical model by Martin is defined as:

> ". . . a model of data which represents the inherent structure of that data and hence is independent of individual applications of the data and also of the software or hardware mechanisms that are employed in representing and using the data (10:235)."

The Canonical model was primarily developed to aid in designing data bases and is as much a process as a model. The model uses bubble charts and is a product of a process called "Canonical Synthesis" (see Appendix H). Due to its incremental nature, the canonical process (model and synthesis) proves to be particularly good at handling dynamic data base instances. The synthesis process is readily automated, although still requiring some amount of human supervision and modification for total effectiveness. The resulting canonical model derived from the synthesis is independent of any model, performance constraints, or particular machine, and is in third normal form (3NF - see Chapter 1). The canon-

ical model as noted earlier uses bubble charts. The bubble chart structures are described below.

Bubble Charts. The most basic piece of data, called an item (also known as field or element) is atomic. Each data item is of a particular type and each type of data item is drawn as a bubble. Each bubble can represent many occurances of that data type (i. e. a name bubble represents many names, not just one).

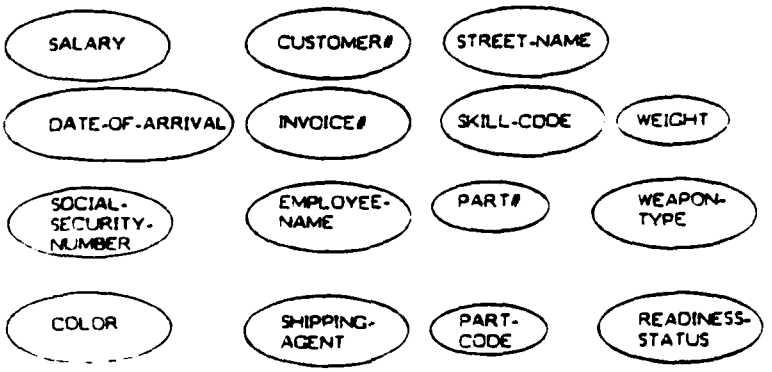


Figure 12. Canonical Bubbles (10:172)

Bubble charts express relationships between bubbles (data items) by connecting arrows which may either be single or double headed. Single headed arrows indicate that a one-to-one relationship (1:1) exists (e. g. each person has one

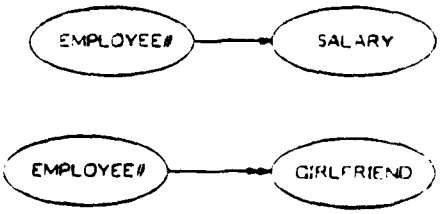


Figure 13. Relation in Canonical Model (10:173)

mother). Double headed arrows indicate a one-to-many (1:M) relationship (e. g. each person can have many friends). A bubble may have any number of arrows entering or leaving it.

In the canonical model these two examples should be combined, although they could be separate.

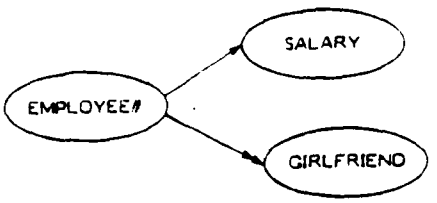


Figure 14. Combining Relationships (10:174)

Further, these expressed relationships need not be only one way but can go both ways, called reverse associations. Reverse associations are not always required and are only stipulated when that particular information is desired.

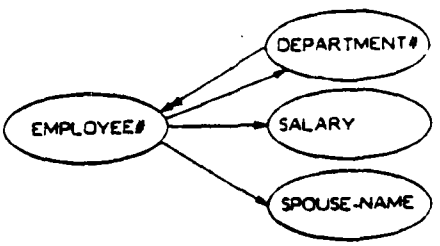


Figure 15. Reverse Associations (10:175)

Bubble charts, and hence the canonical model, permit the expression of 1:1, 1:M, M:1, and M:M relationships. Bubble charts also permit the explicit expression of optional links (relationships). For a value of A there may or may not be a value of B. This is indicated with a zero by the arrowhead.

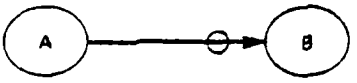


Figure 16. Optional Relationships (10:178)

Bubble charts allow multiple associations between data items by allowing more than one link (arrow) to connect two bubbles. To identify the different relationships, the arrows are labeled.

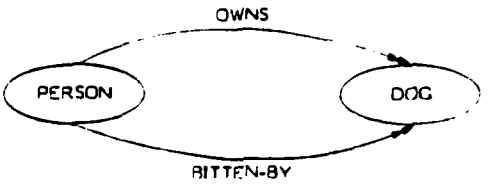


Figure 17. Multiple Relationships (10:185)

This type of situation can be avoided by the inclusion of an extra data type:

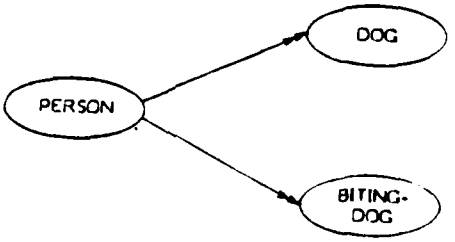


Figure 18. Removing Multiple Relationships (10:185)

Bubbles are also permitted to be linked to themselves,

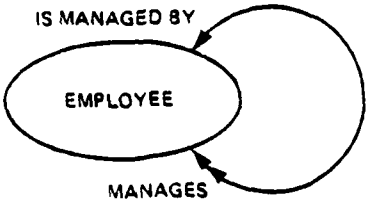


Figure 19. Looping Relationships (10:186)

called looping. As with the multiple associations, the arrows are labeled. The example below shows that some employees can manage other employees.

Generally, a very large portion of the data items are grouped together because it is impractical to handle all of the possible associations between all of the data items in a given data base. These data-item groups are called tuples, records, or segments in other DBMS. A data-item group is drawn as a bar containing the names of the data items it includes:

Figure 20. Data-Item Group (10:176)

The above grouping represents the following bubble chart:

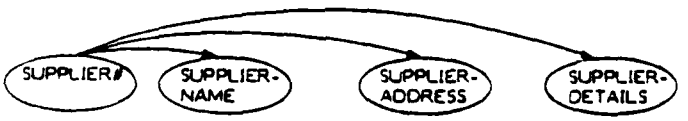


Figure 21. Data-Item group in Bubble Format (10:176)

These data-item groups are then treated as if they were a single bubble.

Each data-item group requires something to uniquely identify that grouping of data-items. This is done by specifying one of the data-items in the group as the key to that grouping. The key is not chosen arbitrarily but rather logically. The data-item group in Figure 16 has the supplier number as the key since, of the four data-items in the group,

it alone can uniquely identify that group. The supplier number can be unique, however the name, address, and details could all possibly be duplicated (i. e. two suppliers could have the same name, or same address, etc.). The following list of definitions describe the concept of keys and attributes (10:177-178):

> Primary key: a bubble with one or more single-headed arrows leaving it. A primary key may identify many data items. In more generic data base terms, a primary key is an attribute (nonprime) which uniquely identifies a tuple (data-item group).
>
> Nonprime attribute: all items that are not primary keys (often just called an attribute). A bubble with no single-headed arrows leaving it.
>
> Secondary key: nonprime attribute with one or more double-headed arrows leaving it. A secondary key does not uniquely identify another data item but rather one value of a secondary key is associated with zero, one, or many values of another data item.
>
> Concatenated key: a composition of one or more data items which uniquely identify a group of data items which cannot be uniquely identified by only one key.

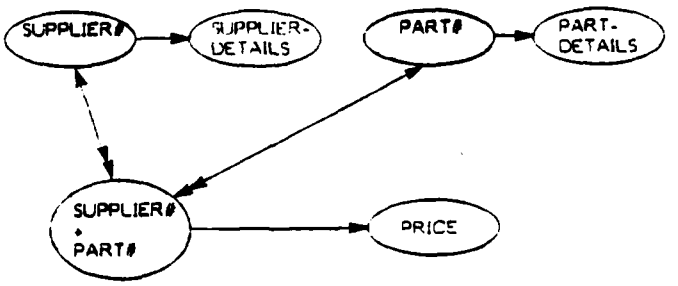


Figure 22. Concatenated Keys (10:184)

A final capability of bubble charts is allowing different levels of primary keys. Sometimes, particularly in tree structure representations of data, some primary keys identify

other primary keys. To improve the clarity of the bubble chart structure, single arrows are often drawn between the primary keys pointing upward whenever possible. The data-item group at the top of the "tree" is called the root key. The root key is "a primary key with no single arrows leaving it to another primary key (10:186)." In the tree example there is only one root key, in network structures there may be several root keys. The primary keys in these structures can be arranged into levels with the highest level, depth 1, being the root key(s). Depth 2 primary keys have a single-arrow link to a depth 1 primary key. Depth 3 primary keys have a single-arrow link to a depth 2 primary key, and so forth.

Canonical Synthesis. As noted previously, the Canonical model is as much a process as a model. The Canonical model is a result of applying the Canonical Synthesis (see Appendix H) on the bubble chart structures just described.

> "The process of canonical synthesis creates the logical model of data . . . This model is then converted into a logical representation (schema or schemas) [for actual implementation of the data base] . . . (10:236)."

The process takes different user's views of the data and gradually incorporates them into the data base, incrementally eliminating redundancies, placing the data base into 3NF, and so forth. This discussion of the canonical synthesis will only involve additional features and construct or capabilities which could influence the canonical model's ability

to represent different models.

Due to the nature of the canonical model (and bubble charts), it is very convenient and easy to incorporate additional views or even different views of the data. This capability is what gives the canonical model its ability to handle dynamic data bases.

In the previous discussion, primary keys where defined as a bubble with one or more single arrows leaving it. However, it does occur in certain situations that one or more data-items (primary keys) might uniquely identify a data-item group. These data-items are referred to as candidate keys. A good example of a candidate key would be an employee data base where both an employee's social security number and an in-house employee number would uniquely identify an employees name, and address (the data-item group being made up of the social security number, employee number, employee name, and employee address).

It is not always the case that data items are associated only with other data-items but also can be associated with an association itself. An example would be a data item PRICE which cannot be associated with only the Part record or only the SUPPLIER record but rather both. Such data is called intersection data (Figure 23).

Another type of intersection is an intersecting attribute which is an attribute attached to more than one primary key (it has more than one single-headed arrow pointing to

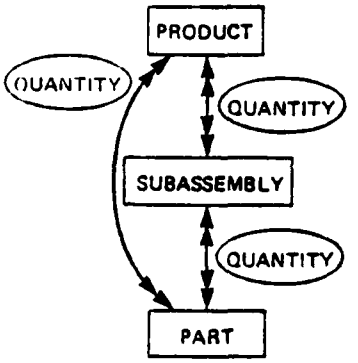
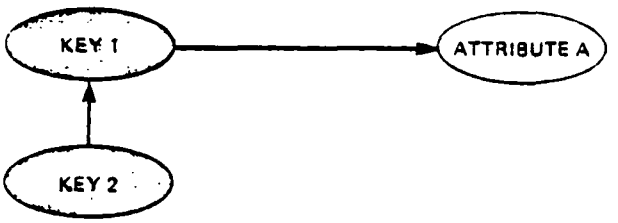


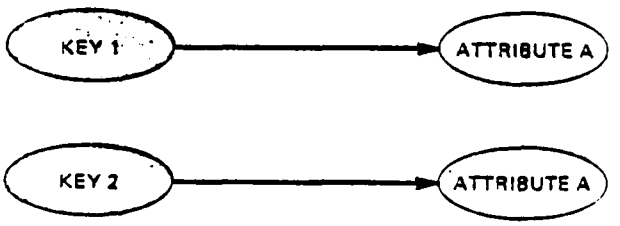
Figure 23. Intersecting Data (10:244)

it). Intersecting attributes are not allowed in the canonical model and can be handled in one of the three ways shown in Figure 24.

1.



2.



3.

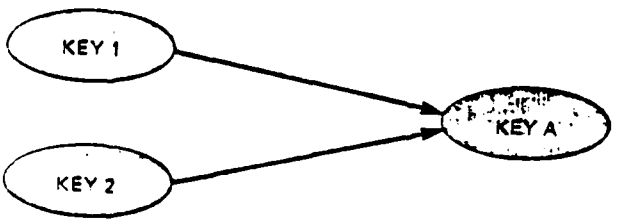


Figure 24. Intersecting Attributes (10:247-248)

The canonical model permits the user to specify 1:1, 1:M, M:1, and M:M relationships. Unfortunately, there are several problems with M:M relationships. First, it is very difficult to physically represent a M:M relationship. 1:M relationships generally use a heirarchical structure or a linked list structure to physically represent the 1:M. Unfortunately, neither of these methods will work with a M:M. The solution for a M:M (A <<--->> B)is to have one file of A records, one file of B records, and one file showing how they are related. The second problem is that generally with M:M associations, intersection data will usually be associated with it sooner or later. For this reason, M:M associations should be avoided in the canonical model (10:245-246).

The canonical model does not allow the user to indicate the sequence in which records are stored.

> "In general, it is not desirable to state a record sequence in the canonical model because different applications of the data might require the records in a different sequence (10:248)."

The canonical model does allow the specification of secondary-key paths. These paths are generally used to increase the speed of searches. Interactive systems often employ secondary-key paths.

Canonical DML. No DML has been defined for the Canonical model since it was never actually designed to be a model for a DBMS. Rather it was designed to help a DBA in analyzing a particular data base intension. If the Canonical model would be chosen for the UDM, a DML would have to be written

for it.

Canonical Conclusion.

> "A canonical database structure is a minimal non-redundant model. Its records are in third normal form (and fourth normal form). . . [and] is sometimes referred to as a 'conceptual schema'. . . A canonical model can be represented as a network (CODASYL), heirarchical (IMS), or relational database system. A large canonical model may be kept, updated, and designed by computer, to represent overall the data which are the foundation of a computerized enterprise (10:275-276)."

Entity-Relationship Model

Entity-Relationship (ER) models are based on tables and graphs and were an outgrowth of the designing of data bases using commercial DBMS. Due to this fact, ER models bear a strong resemblance to the heirarchical and network models. However, the ER models are generalizations of these two models (accomplished by a direct representation of explicit constraint and M:M relationship types).

The ER model to be discussed was proposed by Chen (2) in 1976 and is considered to be probably the best known of the ER models (9:175). Originally, Chen's ER model was designed for the purpose of data base design by allowing the specification of an enterprise schema. An enterprise schema represents an enterprise's view of its data, independent of storage or efficiency considerations. Unlike the other ER models, Chen's ER model's conceptual schema is not necessarily directly accessible by a DBMS. The ER model only documents the logical properties of the data base and may or may

not be directly accessible.

ER Structure Glossary. The following section is a glossary of terms defining the basic structures of the ER model.

> Entity set: Something with objective reality which exists or can be thought of (9:26). Represents the generic structure of an entity in an enterprise's realm of interest (9:176). A thing which can be distinctly identified (2:10).
> Entity key: A set of one or more attributes which uniquely identify an entity set. Artificial attributes may be added to create an entity key satisfying the above conditions.
>
> Relationship set: Represents the generic structure of the relationships among entity sets (9:176). An association among entities (2:10). "Father-Son" is a relationship among two person entities.
>
> Relationship key: Serves same function as the entity key and is composed of the entity keys of the entity sets involved in the relationship set.
>
> Role: Function that an entity performs in a relationship. "Husband" and "Wife" are sample roles (2:12).
>
> Value Set: A domain. Examples are Name, State, Color, and Skills.
>
> Value: A particular instance of some value set. Membership in a value set dependent on a predicate (9:179).
>
> Attribute: A mapping between an entity set or relationship set and a value set (9:179).

ER Structures. The ER model allows the graphical depiction of a data base through an entity-relationship diagram (ERD). The ERD shows the intension of a data base. Figure 25 is an ERD example for a medical data base.

In an ERD, entity sets are represented by rectangular, labeled boxes. Relationship sets are represented by diamond,

labeled boxes. Arcs connect the entity sets which are participating in a particular relationship set. The arcs in the ER model allow the depiction of 1:1 and M:M relationships. Recursive links are also allowed. Mapping properties of a relationship are given explicitly in an ERD but only the maximum cardinality allowed for an entity set in a relationship

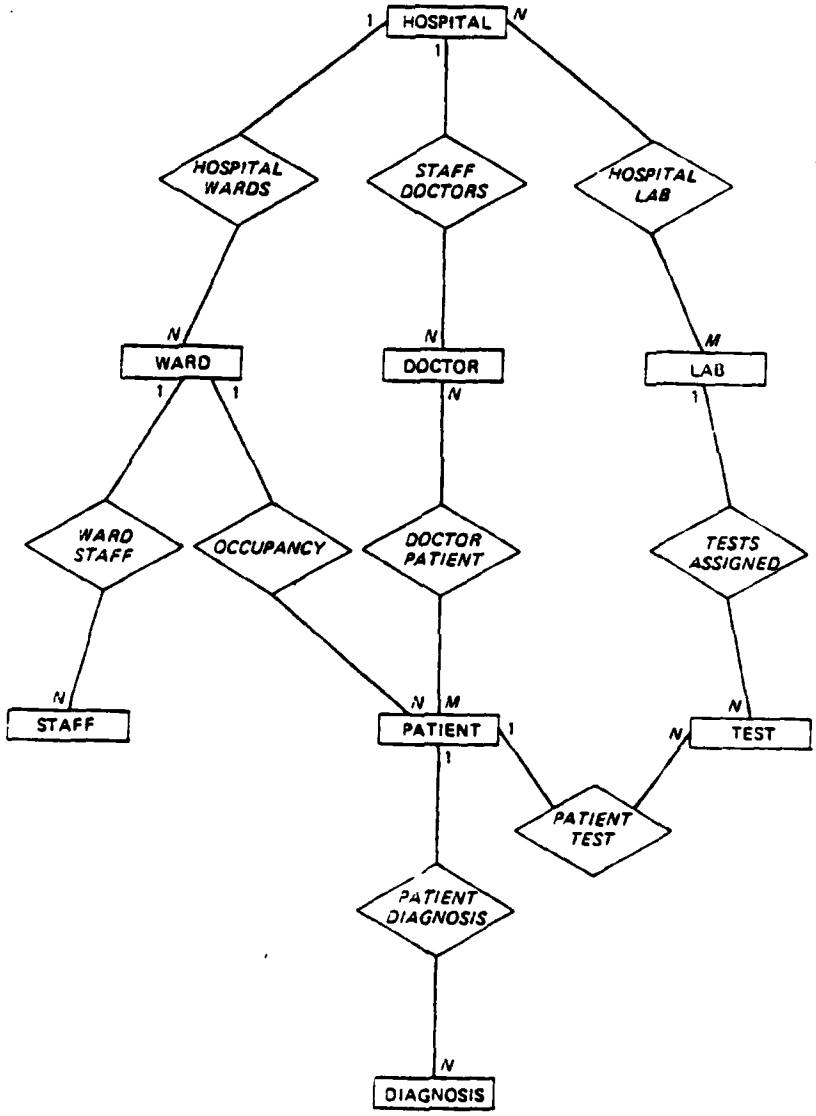


Figure 25. ERD Diagram for Medical Data Base (9:177)

set is indicated (by label on arc). Letters indicate no maximum limit on the cardinality.

The ERD allows the specification of more than one relationship set between the same two entity sets.

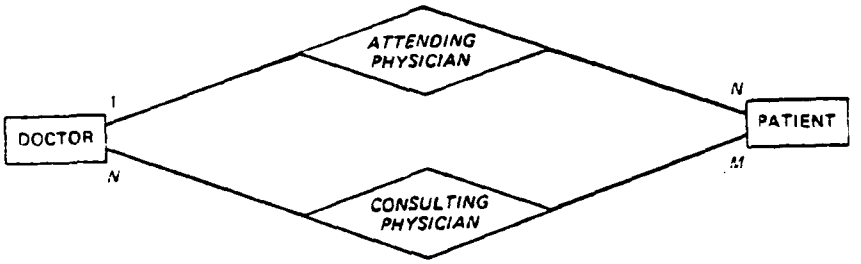


Figure 26. Multiple Relationships (9:178)

It also permits recursive relationship sets and allows relationship sets to have roles which is indicated on the ERD by labeling the arcs.

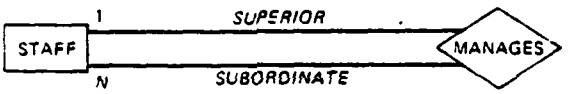


Figure 27. Recursive Relationship Set (9:178)

Besides showing the mappings between entity sets and relationship sets, the ERD also shows attribute mapping by a directed arc from the entity set (Figure 28) or relationship set (Figure 29) to the appropriate value set(s). The ER model permits this while the network and heirarchical models disallow it and the relational requires an extra relation to handle it.

ER Constraints. Constraints in the ER model are almost exclusively explicit. One of the few possible inherent constraints is the requirement that entity set and value set

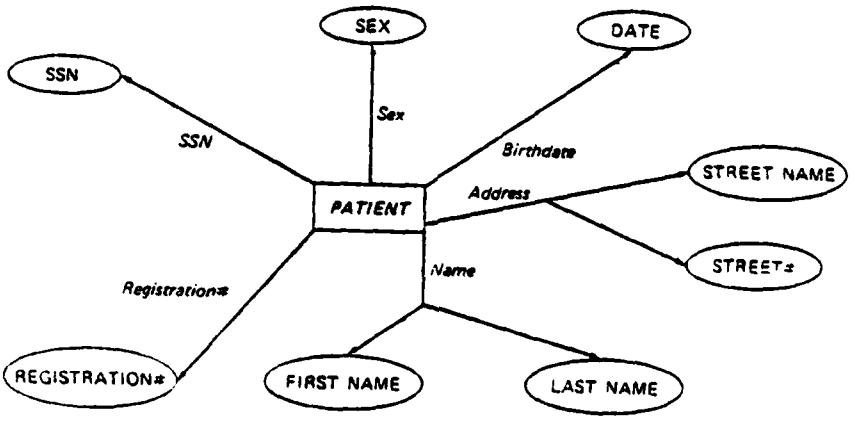


Figure 28. Attributes and Value Sets (9:180)

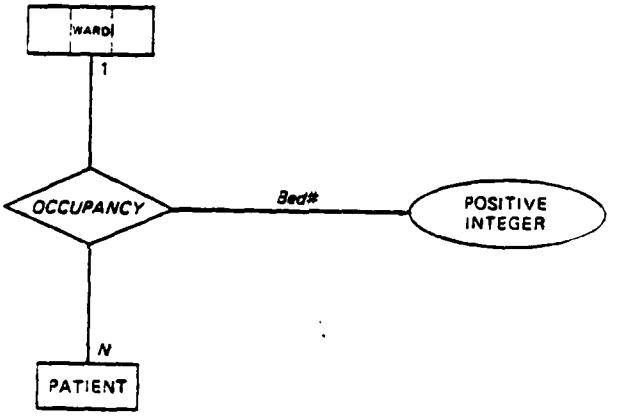


Figure 29. Attribute of a Relationship Set (9:181)

membership be determined by a predicate of some sort. There are several constraints which are explicitly specified:

> Domain Specification: The value set of an attribute specifies the values that the attribute can assume. For example, the Bed# and # of beds in the medical data base could be limited to the value set POSITIVE INTEGER. These could be further restricted by specifying a range, 1 to 100, for the Bed# and # of beds. The ER model can express constraints on existing values in the data base which can be between sets of values or particular values. An example of the former would be the entity sets for DOCTORS and ILL DOCTORS. The entities of ILL

DOCTORS should be a subset of the entity set DOCTORS. The latter would occur when the sum of the # of beds in the wards must equal the sum of the beds in the hospital (9:180).

Existence Constraint (existence dependency): This constraint indicates that the existence of a particular entity set is dependent on the existence of an associated entity set. For example, the existence of the entity WARD is dependent on the existence of the entity HOSPITAL. If the HOSPITAL entity set is deleted, so shall the WARD entity. The ERD represents this with a double-rectangle box with a label E in the associated relationship set box and an arrow pointing to the dependent entity. The dependent entity set is termed a weak entity set and the associated relationship set a weak relationship set. Those sets not termed weak are termed regular.

Figure 30. Existence Constraint (9:182)

ID Dependency: A case where an entity cannot be identified by the value of its own attributes, but rather by its relationship(s) with other entities. The ERD represents this the same way as the existence dependency but the box is labeled with ID rather than E. An ID dependency is automatically an existence constraint but not necessarily vice versa.

ERD Extension. The extension of an ERD is represented by a series of tables called an entity relation (Figure 32). Although strongly resembling a data base relation (see next

Figure 31. ID Dependency (9:183)

section and/or Chapter 1), the entity table relation is not truly a data base relation because of the possibility of weak entity sets and no requirement for an entity key. If the entity relation does have a key than it can be viewed as a data base relation. Each row of the table is called an entity tuple. The extension of an entity set has one column for each attribute associated with the set. The extension of a weak entity set, weak entity relation, consists of all of the

PRIMARY KEY

ATTRIBUTE	EMPLOYEE-NO	NAME		ALTERNATIVE-NAME		AGE
VALUE SET (DOMAIN)	EMPLOYEE-NO	FIRST-NAME	LAST-NAME	FIRST-NAME	LAST-NAME	NO-OF-YEARS
ENTITY (TUPLE)	2566	PETER	JONES	SAM	JONES	25
	3378	MARY	CHEN	BARB	CHEN	23
	⋮	⋮	⋮	⋮	⋮	⋮

RELATIONAL VIEW

Figure 32. Entity Relation (2:17)

attributes of the weak set plus the entity key of the regular

entity set(s) with which it is associated.

The extension of a relationship set is called a relationship relation (Figure 33) and consists of the entity keys of the entity sets associated with the relationship set and, if any, attributes of the relationship set itself. Each row is called a relationship tuple. Weak relationship relations also exist, for weak relationship sets, but since weak entity sets don't have entity keys, they must be identified by their relationship with other entity sets.

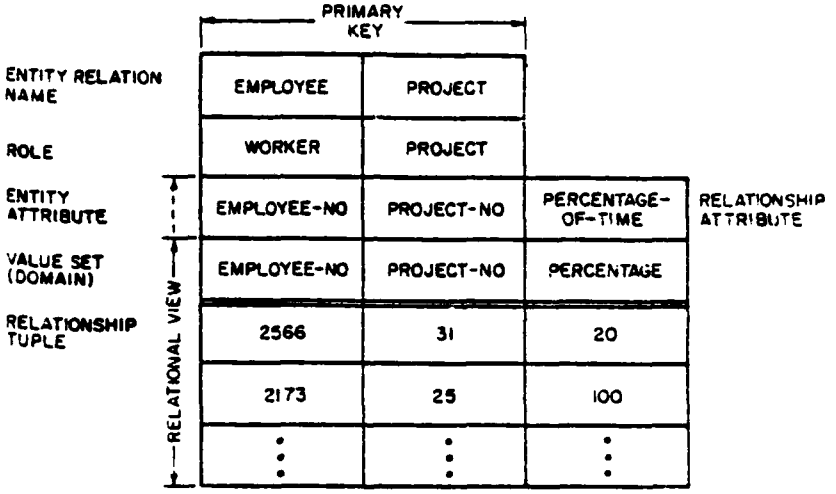


Figure 33. Relationship Relation (2:17)

ER DML. When the ER model was originally defined, no DML was specified but it was indicated that information requests could be expressed using set notions and operations (9:185). In 1978, a DML was specified by Shoshani called CABLE (ChAin-Based LanguagE).

> "The language specification concentrates on output and selection and is based on the concepts of chains , or paths, through the entities and relationships of the data base. The language makes use of the fact that usually not all elements of a chain need to be specified, but can be inferred from the

schema. . . A chain or path in the data base is composed of elements called beads. Each bead is an elementary selection criteria on either an entity set or a relationship set. The syntax of a bead is

$$\left\{\begin{array}{l}\text{entity set name}\\ \text{relationship set name}\end{array}\right\}\text{.qualification}$$

. . . Both the qualification and the entity set name or relationship set name can be optional in a bead. If the attribute names are unique in a schema, the entity set name or relationship set name can be omitted. If the bead is not qualified , the entity set name or relationship set name serves merely to specify the path in the data base. Beads in a path can be omitted if no ambiguity arises (9:186)."

Examples (9:186-187):

List the names of all doctors whose specialty is gynecology.

```
SELECT DOCTOR. Specialty= 'GYNECOLOGY'
```

List the names of the patients in hospital 22. Since several paths exist from the HOSPITAL entity set to the PATIENT entity set, the path (WARD) must be specified.

```
OUTPUT PATIENT. Name
SELECT HOSPITAL. Hospital code = '22'/ WARD
```

List the name and specialty of doctors treating patients who have tests being performed by lab 86 and who are in hospital 22. For this query there are several different possible paths, therefore, one must be specified.

```
OUTPUT DOCTOR. Name, Specialty
SELECT HOSPITAL. Hospital code = '22' /
   LAB. Lab# = '86' /
   PATIENT
```

ER Conclusion. The ER model is a flexible and powerful model. It possesses capabilities which allow it to express a

variety of data base structures and instances. Due to its logical nature, it can express more than a commercial DBMS.

The Relational Model

The relational model was originally proposed by Codd and arose out of a desire to bring some sort of formalism in addressing various issues and problems in the area of data base design. The obvious answer was to use already formulated mathematical theory. The following three observations/definitions provide a good idea of what the relational model is. The section following this (relational structure glossary) will provide a list of definitions of the structures used in the relational model. The relational operators section will describe the general operations allowed on a relational model. They are included in the discussion of this model because the model is based on mathematical relation theory and those operations are an integral part of the that theory. The next section describes the constraints that the relational model places on the data within the model.

> "The relational approach to data is based on the realization that files that obey certain constraints may be considered as mathematical relations, and hence that elementary relation theory may be brought to bear on various practical problems of dealing with data in such files (3:65)."

> "Definition: Given a collection of sets D1, D2, . . ., Dn (not necessarily distinct), R is a relation on those n sets if it is a set of ordered n-tuples <d1, d2, . . ., dn> such that d1 belongs to D1, d2 belongs to D2, . . ., dn belongs to Dn. Sets D1, D2, D3 are the domains of R. The value n is the degree of R (3:83)."

"The relational model, as defined by Codd [COD82], consists of three basic parts, a collection of relations that describe the logical structure of the database, a collection of operators to manipulate data stored in the database, and a collection of general integrity rules that constrain the set of valid states of the database (8:47-48)."

Relational Structure Glossary.

Relational Extension: "the set of tuples appearing in a relation at any given instant (3:90)." Also known as a view.

PART

P#	PNAME	COLOR	WEIGHT	CITY
P1	Nut	Red	12	London
P2	Bolt	Green	17	Paris
P3	Screw	Blue	17	Rome
P4	Screw	Red	14	London
P5	Cam	Blue	12	Paris
P6	Cog	Red	19	London

Figure 34. Example Relation (3:83)

Relational Intension: "[in contrast to the extension, the intension] is independent of time. Basically, it is the permanent part of the relation; in other words, it corresponds to what is specified in the relational schema (3:90)." See Figure 35.

```
DOMAIN   S#       CHARACTER (5)  PRIMARY
DOMAIN   SNAME    CHARACTER (20)
DOMAIN   STATUS   NUMERIC   (3)
DOMAIN   CITY     CHARACTER (15)
DOMAIN   P#       CHARACTER (6)  PRIMARY
DOMAIN   PNAME    CHARACTER (20)
DOMAIN   COLOR    CHARACTER (6)
DOMAIN   WEIGHT   NUMERIC   (4)
DOMAIN   QTY      NUMERIC   (5)

RELATION  S  (S#,SNAME,STATUS,CITY)
             PRIMARY KEY   (S#)
             ALTERNATE KEY(SNAME)
RELATION  P  (P#,PNAME,COLOR,WEIGHT,CITY)
             PRIMARY KEY   (P#)
RELATION  SP (S#,P#,QTY)
             PRIMARY KEY   (S#,P#)
```

Figure 35. Relational Schema for Suppliers-and-Parts Data Base (3:92)

Domain: Consists of a domain name, unique to the data base, and a fixed, non-empty set of domain values that describe the possible values for a given attribute (8:48). A pool of values from which the actual values of an attribute can be drawn (3:65). Example would be COLOR in figure 34.

Primary Domain: Any domain for which there exists some single-attribute primary key.

Attribute: Consists of an attribute name, unique to the relation, and a domain (8:48). A column in a table (relation) which represents some atomic piece of information. Example, from figure 34, would be that the domain COLOR has the attributes Red, Green, Blue, and Red.

Relation: A table-like structure that consists of a relation name, a non-empty set of attributes, and a time varying set of tuples (8:48). See figure 34.

Tuple: A row in a relational table. Often thought of as a record of information which contains various attributes and is uniquely identified by some key. One example, from figure 34, would be the row where P# = 1.

Primary key: An attribute whose distinct value will uniquely identify a given tuple from all other tuples in the relation. Example would be P# from the relation PART.

Foreign key: A primary key in a relation for which it is not the primary key for that relation (but is for some other relation).

Candidate key: A non empty set of attributes belonging to a relation which uniquely identify, on a one-to-one basis, each tuple in the relation (8:48). A set of attributes possessing the unique identification property (3:88).

Alternate key: A candidate key that is not a primary key.

Relational Operators. This section describes the relational algebra operations that are part of the relational model. Figure 36 is provided for reference in understanding

the operators.

S

S#	SNAME	STATUS	CITY
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

SP

S#	P#	QTY
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P2	200
S4	P4	300
S4	P5	400

P

P#	PNAME	COLOR	WEIGHT	CITY
P1	Nut	Red	12	London
P2	Bolt	Green	17	Paris
P3	Screw	Blue	17	Rome
P4	Screw	Red	14	London
P5	Cam	Blue	12	Paris
P6	Cog	Red	19	London

Figure 36. Suppliers-and-Parts Data Base (3:92)

Union: The union of two (union-compatible) relations A and B, A UNION B, is the set of all tuples t belonging to either A or B (or both) (3:205).

Intersection: The intersection of two (union-compatible relations A and B, A INTERSECT B, is the set of all tuples belonging to both A and B (3:205-206).

Difference: The difference between two (union-compatible) relations A and B, A MINUS B, is the set of all tuple t belonging to A and not to B.

Extended Cartesian Product: The extended Cartesian product of two relations A and B, A TIMES B, is the set of all tuples t such that t is the concatenation of a tuple a belonging to A and b belonging to B (3:206).

Selection: The algebraic selection operator . . . yields a 'horizontal' subset of a given relation - that is, that subset of tuples within the given relation for which a specified predicate is satisfied (3:208).

Projection: The projection operator yields a 'vertical' subset of a given relation - that is, that subset obtained by selecting specified attri-

butes, in a specified left-to-right order, and then eliminating duplicate tuples within the attribute elected (3:208).

S WHERE CITY = 'LONDON'

S#	SNAME	STATUS	CITY
S1	Smith	20	London
S4	Clark	20	London

Figure 37. Sample Selection (3:208)

S[CITY]

CITY
London
Paris
Athens

S[SNAME, CITY, S#, STATUS]

SNAME	CITY	S#	STATUS
Smith	London	S1	20
Jones	Paris	S2	10
Blake	Paris	S3	30
Clark	London	S4	20
Adams	Athens	S5	30

Figure 38. Sample Projection (3:209)

Join (Natural): If two tables have a column defined over a common domain, they may be joined with the resulting table being wider in which each row is the concatenation of the two being joined. This is called an equijoin. When one of the two identical columns is eliminated, that join is called a natural join (3:76).

S.S#	S.SNAME	S.STATUS	S.CITY	SP.P#	SP.QTY
S1	Smith	20	London	P1	300
S1	Smith	20	London	P2	200
S1	Smith	20	London	P3	400
S1	Smith	20	London	P4	200
S1	Smith	20	London	P5	100
S1	Smith	20	London	P6	100
S2	Jones	10	Paris	P1	300
S2	Jones	10	Paris	P2	400
S3	Blake	30	Paris	P2	200
S4	Clark	20	London	P2	200
S4	Clark	20	London	P4	300
S4	Clark	20	London	P5	400

Figure 39. Sample Join between S and SP (3:210)

Division: The division operator divides a dividend relation A of degree m + n by a divisor relation B of degree n, and produces a result relation of de-

gree m. . . . . the result of dividing A by B - that is, A DIVIDEBY B - is the set of values x such that the pair <x,y> appears in A for all values y appearing in B. The attributes of the result have the same qualified names as the first m attributes of A (3:211).

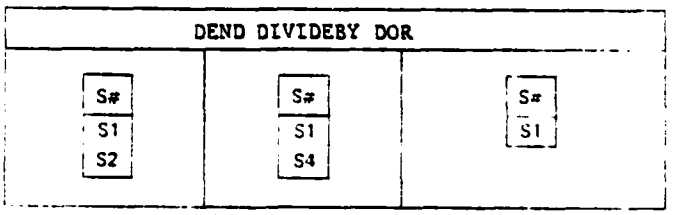


Figure 40. Sample Division (3:211)

It may not be apparent from these definitions exactly how a query is processed, therefore, let us examine a sample query and indicate how the operations are applied (optimization will not be considered). Query: What suppliers in London supply at least 200 units of P2? One of many possible ways of handling this query would be to do a join of the S (supplier) and SP (supplier-part) tables, S JOIN SP (see figure 39), forming a temporary table called SSP. Next, select from SSP where the S.CITY = 'London', SP.P = 'P2', and where the SP.QTY >= 200 (SSP WHERE S.CITY = 'LONDON' AND SP.P = P2 AND SP.QTY = 200). This results in a table with two entries (second and tenth lines of the table in figure 37). Since the query desired only the supplier's names, S.NAME is projected out resulting in a single column table with Smith and Clark as the two attributes (and the answer).

Relational Integrity Constraints. In any data base situation there is a need to insure that the data contained

within the data base conforms to some definition of correctness (8:58). This can be accomplish, to some extent, by establishing constraints, Integrity constraints, on how the data can be stored in the data base. The first two constraints listed are required to have a complete relational data base (8:59). Others have also been proposed and are listed.

> Entity Integrity (Integrity Rule 1): No part of a primary key may be null. Note: partially null identifiers should also be prohibited (3:89).
>
> Referential Integrity (Integrity Rule 2): "A foreign key must have a value that is in the primary key or be null (8:60)."
>
> Domain Integrity: "identifies the characteristics of data that can be stored in a given attribute (8:62)." Insures that the correct type of data is stored in a given attribute.
>
> Immediate Record State Constraint: a constraint placed on an individual attribute value in terms of the range of the data values (8:61). For example, the attribute AGE might be constrained to range only from 0 to 120.
>
> Immediate Record Transition Constraint: a constraint placed on changing a value relative to its current value. For example, the attribute TRANSACTION_DATE might be constrained that the NEW_TRANSACTION_DATE > TRANSACTION_DATE (8:62).

Relational DML. The relational operators previously described form the underlying base for most relational DMLs. Most relational DBMSs, like System R, develop a DML with a user friendly frontend DML which is then translated into relational algebra and evaluated. The language specified in Chapter 7 is an example of this type of setup.

Relational Conclusion. The relational model is a very simple and powerful model which has proved to be very popular in recent years. In this description of the relational model, the emphasis has been on providing the structures, operators, and constraints that constitute the relational model. For a more tutorial description of the relational model, the reader is asked to read Chapter 1 again which has such a description of the relational model.

Conclusion

In this chapter, the three candidates for the UDM have been described. A conscious effort has been made to describe the models in a general application context. In Chapter 5, the three models will be evaluated in the context of the UDB and the objectives developed in Chapters 1 and 2.

V. The Universal Model

Introduction and Overview

The first four chapters of this thesis have been directed towards a discussion of the UDB problem, some goals or objectives to be satisfied in solving that problem, and an analysis of the system that needs to be designed. The last chapter briefly described three models which were being considered for the universal model. In this chapter, these three models are evaluated in terms of the UDB application. Each is analyzed and its strengths and weaknesses noted for this application. Then a set of selection criteria is established and applied against the three models (comparatively). One model is chosen as the UDM. The conclusion of this chapter then evaluates the effect that the particular model chosen has on the observations and design objectives described in Chapter 2. A final set of design objectives/goals is then established.

The Universal Data Base Context

In this section of Chapter five, each model is evaluated for its abilities in the UDB application.

Relational Model. The relational model possesses a strong capability in a UDB application. The relational model is a proven model, its capabilities are known (although debated). The necessary mapping algorithms to the other two models have already been developed, or at least examined.

The relational model is very simple and easy to learn and use. A summary of the relational model's strengths are listed below:

RS1. Simple and easy to learn and use.

The key to this strength is the fact that the relational model is not only easy to use but also easy to learn. The latter factor will prove especially useful if it is decided that the users will have to work in the universal model and language.

RS2. Table format lends itself well to a distributed environment.

This is, perhaps, the most important strength for the relational model. The table format of the relational model lends itself particularly well to the UDB application. It aides in presenting unrelated information and additional information since these can be handled simply by the addition of more tables (relations).

RS3. A well established model. Its capabilities are known as well as its weaknesses. Many of the mappings (relational to DBTG, etc.) have been done or at least examined.

Obviously, having an established model will make the implementation and acceptance of this model somewhat easier. There would be some savings in terms of the fact that the relational to relational mappings would be easier and the relational users would have to make a minimal adjustment. Introducing a relatively unknown model would involve more work and research then with this model.

RS4. The relational model shields the user from the underlying data formats and complexity of the data structures.

The primary value of this strength, in the UDB application, is the reduction in complexity.

RS5. It has a high level, nonprocedural DML which has proved to be easier to use and more productive for programmers (6:4).

Although the design objectives in Chapter 2 stated that the UDML should support both procedural and nonprocedural operations, it is important to note that it may not be practical nor preferably to support both. Nonprocedural languages are generally easier to use and more productive for the programmers. Furthermore, it may not be possible to support navigational operations in a distributed environment.

RS6. Storage and data structures are very simple (6:4).

A minor strength in this particular application.

RS7. Access paths don't have to be predefined (contrary to procedural languages/models) (6:4).

This is a particularly good feature of the relational model. Besides increasing the flexibility and power of the model, this feature will prove extremely beneficial in a distributed environment. See Chapter 2, Design Objective 19.

RS8. The relational model has a fast response time to ad hoc queries which are considered to be a high-percentage of the queries submitted (6:5).

Certainly a strength for a regular DBMS, and still to some extent the UDBMS, but its impact in the UDBMS will not be very significant since the model would only be functioning

as a communication media. It would not be actually processing queries (the LDBMSs would be).

RS9. The relational model handles M:M relationships extremely well.

As pointed out earlier (Chapter 4), M:M relationships can prove difficult to actually implement. The relational model handles this situation quite well because the M:M relationships are not physically stored by a linked structure like those in most heirarchical or network DBMS. M:M relationships in the relational model are logically stored not physically.

RS10. The relational DML is highly parseable and well suited to optimization.

This is another strength which will be very valuable in the UDB application due to the distributed nature of the application.

The weaknesses of the relational model are summarized below:

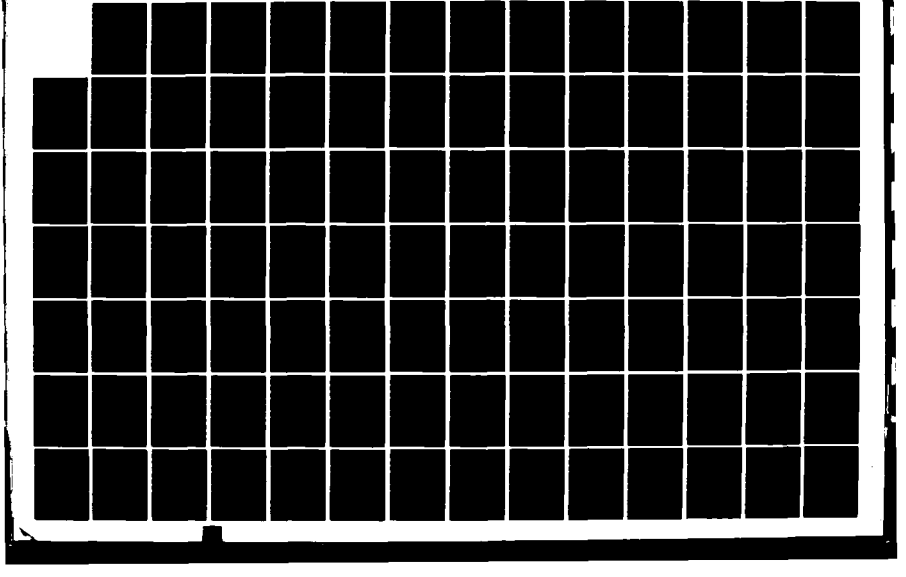
RW1. The relational model is one which has been implemented and, therefore, has taken into consideration machine efficiency, etc. Perhaps, since the desired model need only appear to be a real DBMS, a purely logical model (canonical or ER) might be more flexible or better suited to the UDM role.

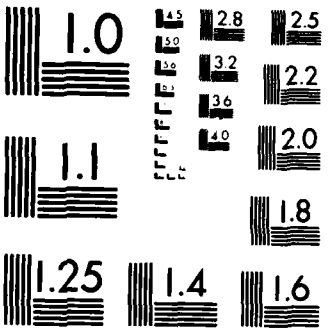
RW2. Even though the relational model (and operations) can be mapped to the network and heirarchical models (nonprocedural to procedural), perhaps it would be more efficient to have a model with the built-in ability to do procedural operations rather than just being mapped into them.

RW3. The relational model cannot convey procedural operations.

AD-A151 856 ANALYSIS AND SPECIFICATION OF A UNIVERSAL DATA MODEL FOR DISTRIBUTED DATA..(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. A J JONES 2/4

UNCLASSIFIED 14 DEC 84 AFIT/GCS/ENG/84D-11 F/G 9/2 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

If the relational model is chosen as the UDM, universal procedural operations will not be supported. Obviously, procedural operations could be mapped into the relational model but this would done taking into consideration the entire query. Essentially, a user could not single step through a distributed data base without extreme difficulty in most cases and almost impossible in others.

RW3. In general, perhaps a better approach would be to tailor the UDM to its environment rather than trying to fit the relational model into that role.

RW4. All constraints are not explicit.

See Chapter 3, Data Model Mapping, for a discussion of the benefits of the explicit expression of constraints.

The Entity-Relationship Model. The ER model also possesses good potential as the UDM but in different way. The ER model is not as simple as the relational (or canonical) but has the capability to inherently represent all three of the models required for the UDB (relational, network, and heirarchical). A summary of the ER model's strengths are listed below:

ERS1: The ER model is a generalization of the network and heirarchical model and, therefore, will prove easier for users of similiar systems to learn and should assist in the mapping process. Furthermore, the ER model can be extended into a pseudo-relational format which could be modified to make it a legitimate relational model.

This is certainly the ER model's strongest benefit. This particular strength would assist in the mappings and in presenting the UDB to the individual local users in a format

similiar to the local format.

ERS2: The ER model can support both procedural and non-procedural operations.

This particular strength is one of the ER's best. The ER model would have much more power and flexibility in supporting procedural operations in the distributed environment. It would still not be able to support all facets of such operations.

ERS3: The ER model, while based on two real world models, is a logical model and, therefore, may present the best of both worlds.

ERS4: The ER explicitly states all constraints.

See Chapter 3, Data Model Mapping, for a discussion of the benefits of explicit expression of constraints.

ERS5: Since the ER model can represent all three of the models, it will prove easier to represent the information to all users in a format similiar to their local model.

The ER model's weaknesses are listed below:

ERW1: More complex than the other two models at the user interface level.

ERW2: A more complex DML required to support procedural and nonprocedural commands.

ERW3: Relationships must be predefined.

See Chapter 2, Design Objective 19, for a discussion of predefined versus non-predefined relationships.

The Canonical Model. The canonical model, much like the ER model, is a generalization of the network model, and to lesser extent the heirarchical. Once the data-items are grouped together, the canonical model begins to resemble a

network model. Fortunately, the canonical model does not have to concern itself with actual implementation, and therefore, some of the complexity involved in such an implementation. The strengths of the canonical model are listed below:

CS1. In general, a simple representation of information.

CS2. Canonical model, and synthesis, provide good support for the different views that would be encountered in a distributed environment.

CS3. The canonical model can support both procedural and nonprocedural operations.

The weaknesses of the canonical model are listed below:

CW1. Access paths and relationships must be predefined.

See Chapter 2, Design Objective 19, for a discussion of predefined versus non-predefined relationships.

CW2. Storage structures are complicated by the fact there is a distinction between entity and relationships.

CW3. All constraints are not explicitly stated.

See Chapter 3, Data Model Mapping, for a discussion of constraints.

Selection Criteria

This section of Chapter 5 describes a set of criteria for use in comparing the three models.

Criteria #1: Simplicity and User Friendliness.

Criteria #2: Ability to depict all three models.

This particular criteria indicates how well the model can present the data base in the three different models. It

will determine if the users will work in their local or the universal model.

Criteria #3: Ability to handle nonprocedural operations.

Criteria #4: Ability to handle procedural operations.

Criteria #5: Implementation benefits.

This criteria evaluates how much of a benefit will be derived, in terms of implementation, from choosing this model. This criteria is best shown by the relational model which has already been implemented with many of the "bugs" worked out.

Criteria #6: Ability to function in distributed environment.

This criteria evaluates how well the model handles the problems of distributed information being represented and manipulated.

Criteria #7: Ability to represent different relationships.

This criteria centers on how well the model represents the different relationships (1:1, 1:M, M:M).

Criteria #8: Flexibility in specification of relationships.

This criteria centers on how whether or not the model has predefined or non-predefined relationships. Chapter 2, Design Objective 16, provides a discussion of this issue.

Criteria #9: Ability to incorporate different user views.

Criteria #10: Ability to support all DBMS functions.

Criteria #11: Ability to express constraints.

This criteria evaluates how explicit the constraints are in each model and how many different constraints can be ex-

pressed. See Chapter 3, Data Model Mappings, for a discussion of constraints.

Application of Criteria

In this section the criteria just described are weighted and applied against each of the UDM candidates. The models are rated comparatively for each criteria (see Tables II and III). The model which best satisfies that criteria receives a three, the second best a two, and third best a one. If two models tie, they are both given the same score. Each criteria is weighted on a scale of one to five with five being the most weight. The criteria are weighted according to which are more important. The more important the criteria, the more weight it is given.

The Universal Model

An examination of Table III brings about the unfortunate observation that the relational model and ER model have tied for the honor of being the UDM. Obviously, only one model can be used for this thesis. Therefore, which one? Both the relational and the ER models offer certain advantages although they both come out equal in overall terms. To break this tie, criteria #3 is removed from consideration. The justification for this is the fact that the support of procedural operations in the distributed environment is impractical to implement. This brings the point totals to 67 (relational), 61 (ER), and 46 (canonical). Therefore, the

relational model is chosen as the UDM.

Table II. Summary of Criteria

1. Simplicity and User Friendliness
2. Ability to depict all three models.
3. Ability to handle nonprocedural operations.
4. Ability to handle procedural operations.
5. Implementation benefits.
6. Ability to function in distributed environment.
7. Ability to represent different relationships.
8. Flexibility in specification of relationships.
9. Ability to incorporate different views.
10. Ability to support all DBMS functions.
11. Ability to express constraints.

Table III. Comparative Evaluation of Three UDM Candidates

Criteria	Weight	Relational	ER	Canonical
1	2	3	2	2
2	4	1	3	2
3	3	3	2	1
4	3	1	3	3
5	1	3	2	2
6	5	3	2	1
7	2	3	3	3
8	3	3	2	1
9	3	2	1	3
10	1	3	3	3
11	3	2	3	1
Total	30	70	70	55

Sensitivity Analysis

The purpose of this section is to investigate how sensitive the results of the evaluation weighting is to change. The approach to this analysis is to choose the two most significant (#2 and #6) criteria and evaluate the resulting change in the final totals from changing the weight factor in each up (a) or down (b) by one. This will be done with all eleven criteria (Start) and with criteria #4 removed (Minus). The table below depicts the resulting changes:

Table IV. Sensitivity Analysis Results

Crit. #	Weight	Rel. Total	ER Total	Canon. Total
Start	30	70	70	55
2a	5	71	73	57
2b	3	69	67	53
6b	4	67	68	54
2a,6b	9	68	71	56
2b,6b	7	67	65	52
Minus 4	27	67	61	46
2a	5	68	64	48
2b	3	66	58	44
6b	4	64	59	45
2a,6b	9	65	62	47
2b,6b	7	64	56	43

An examination of the above results indicates that the original criteria set (criteria #4 included) was very sensi-

tive to changes in the weight factors. The second criteria set (criteria #4 removed) is much less sensitive to these changes and clearly shows the relational model to be the better choice as the universal model (according to the given criteria and weights).

Final Design Decisions

In the course of this thesis, many issues have been raised about the environment, users, and so forth. The way in which these issues were to be resolved was to a great extent dependent on the model chosen for the UDM. Therefore, having made that decision, the following final design decisions are presented. Many of these issues were discussed in Chapter 2 and the reader is directed to that chapter for a more detailed explanation of the reasoning behind each.

> Design Decision 1: Local users will be required to utilize the UDBMS for queries which involve global data. Local users will still be able to utilize their local model/DML for "local only" queries.
>
> Design Decision 2: The data contained within the global system will be presented to the user in a relational format.
>
> Design Decision 3: The UDB, when evaluating any query in the local DML, will notify the user if and when there is additional data in the UDBMS for that local query.
>
> Design Decision 4: Procedural operations will not be supported in the UDBMS. Procedural operations will be mapped from the UDML to those LDBMS supporting procedural operations but the user may not write UDML commands which "navigate" through the UDB.

Design Decision 5: The UDBMS will be a relationally based language but not necessarily any presently designed system.

Design Decision 6: The UDML will support embedded and interactive capabilities which are syntactically similiar.

Design Decision 7: Direct Reference will be supported.

Design Decision 8: Null values will be supported provided that those values are not primary key values.

Design Decision 9: The UDML will have separate constructs for selection and action specification.

Design Decision 10: The UDML will support selection nesting.

Design Decision 11: The UDML will support a form of record-at-a-time capability. The records will be acquired a set-at-a-time but may be analyzed individually in the action specification portion of a query.

VI. The UDDL Mappings

Introduction and Overview

Although the UDM has been chosen through an evaluative process, it has not been shown that the UDM (more specifically the UDDL and UDML) can perform the necessary mappings to the other three models. The primary purpose of this chapter is to examine the UDDL <--> LDDL mapping and related issues. The first section of this chapter discusses various DML mapping issues which might affect the manner in which the DDL mappings are performed. The second section examines mapping issues directly related to the UDDL --> LDDL and LDDL --> UDDL mappings. The third and fourth sections list the restrictions that this thesis effort is placing upon the IMS and DBTG models, respectively. These restrictions are derived from the first two sections of this chapter. The fifth section presents the actual algorithms which were developed to generate the universal representations of the local data bases. The discussions around these algorithms do not constitute a formal proof. The sixth section discusses the integration of the individual representations into one user representation.

DML Mapping Issues

This section of Chapter 6 addresses issues of importance which arose in examining the DML mappings from the UDML to the respective DML's of the DBTG, relational, and IMS data

base systems.

Distributed Information. The issue of distributed information in the DML mapping context centers on the fact that the UDB will have to parse possibly complex UDML commands in a complex UDB environment. The relational nature of the UDML and its modular and parseable design should make this job somewhat easier. This discussion breaks down into three areas: retrievals, updates, and deletions.

Beyond the actual parsing of a UDML query, a complex issue in itself, the difficulty in handling retrievals centers on the question of how and where will the relational operations be done? The initial observations about the UDB system (see Chapter 2) stated that the UDB/UDML would only appear to be a real DBMS and would not actually have any real relational power. Unfortunately, a problem arises, how and where will the results of a parsed distributed query be processed? A query is submitted to a LDBMS which translates it and parses it into the appropriate subqueries. These subqueries are then sent out through the network to the appropriate LDBMSs to be processed. In processing these subqueries the LDBMS will have to translate from the UDML to the local DML to perform the necessary, now emulated, relational operations. These translations are not the problem. The problem arises when operations (i. e. join, selection, etc.) must be done between the intermediate results of those individual subqueries. There are two possible approaches to this

problem. The first approach is to have the originating LDBMS responsible for the final processing. The intermediate results are transferred through the network to the calling LDBMS where the data is placed into existing model and the final processing of the query done.

The second approach is to provide the UDBMS with real relational power. This is accomplished by providing the UDBAC with a data base computer. The intermediate results are now routed to the UDBAC computer, processed, and the final results transferred to the originating LDBMS.

In comparison, both approaches require one transfer of the intermediate data across the network but the second approach has to transfer the final result to the originating LDBMS. The first approach requires that the intermediate data (in a relational format) be placed into the local model/ data base and the UDML query (relational) be translated into the LDML and processed. The second approach requires that the intermediate data be placed into the UDBAC computer but no model or DML translations are required. Further, the processing of the query will be more efficient since the relational operations will be done in a relational manner and not a translated one. Finally, once a query has finished processing, the first approach requires that the original raw data, transferred in from the other sites, be removed from the LDBMS and any added structures be removed. The second approach could merely delete the entire data base which was

established to process the query. A problem the second approach does have is that the UDBAC computer could form a bottleneck in the UDB. However, this problem can be solved with traditional approaches such as multiprocessing, using other relational systems on the network to relieve the UDBAC workload, and so forth. It should be emphasized that the UDBAC computer will be a data base computer and designed for a heterogeneous, distributed network. Therefore, the second approach is advocated by this thesis as the best approach but not without trade offs.

In the context of distributed information (excluding the issue of redundant data) updates have the same complexity and are handled in the same manner as retrievals and, therefore, require no additional discussion. Deletions do pose somewhat of a problem. However, the problem does not really concern the translation of any DML commands but rather the policy towards deletions. This is discussed briefly in Chapter 8.

Redundant Data. While the issue of distributed information presents some difficulties, the issue of redundant data provides even more. Once again the discussion is broken down into three cases: retrievals, updates, and deletions.

Redundant data affects retrievals in terms of the duplicated data being processed. For example, a doctor works for several different hospitals and each has information about that doctor in their LDBMS. Assuming that a UDB user requested information about all of the doctors in the UDB, then that

doctors information (plus any other information similiarly replicated) would be processed at each local site. Unfortunately, there is no real way to prevent that extra processing unless redundant information is not allowed or only completely replicated data is allowed. If partial replication (i. e. a data element or elements may be stored in more than one DBMS. An example would be a person's name and address repeated several different places within a local data base and/or a UDB) the partially duplicated is allowed then the only thing that can be done is to use natural joins at the UDB level (or some other site where the various subqueries are being integrated) to remove the duplicate tuples of information. If no duplicate information is allowed, then redundant data presents no difficulties. If only fully replicated data is allowed, then any queries affecting that replicated information need only be sent to one site where the information has been replicated.

The way updates are handled depends on the which redundant data policy (of those just described) is being used. If partially replicated data is allowed then the update query will have to evaluated so that the affected data at each site will be correctly updated. This could prove to be an extremely complex evaluation. If no replicated data is allowed, then an update query will only have to process the query against the site where that information is stored with no more complexity than that of a retrieval. If only fully

replicated data is allowed then the process is the same as with no replicated data except that each site must be updated.

The approaches to handling deletions parallels the approaches to handling updates. Deletions are more complex in the partial redundancy case. The added complexity arises because deleting a tuple in one particular relation may result in more than only that particular bit of information being deleted in the underlying data bases (i. e. the DBTG and IMS).

The reader is referred to the DDL mapping issues section for a more detailed discussion of the different replicated data policies.

DBTG Set Selection. In the DBTG network model, there often exist situations in which the DBMS needs to select a particular occurrence of a set automatically. To permit this to occur, the DBA must define a SET SELECTION clause within the member subentry of the set entry (3:415). There are three types of SET SELECTION clauses: BY APPLICATION, BY VALUE, and BY STRUCTURAL. The BY APPLICATION type merely indicates that the user specifies the correct occurrence of a set into which to store a new occurrence of the member. The BY VALUE type of set selection clause states that a particular attribute of the owner is used to select the correct occurrence. The final set selection clause type, BY STRUCTURAL, not only specifies the attribute but states that the

value of the attribute in the owner must equal the value of the attribute in the member.

Once again, the question arises as to how the UDDL will handle these DBTG specifications. The BY APPLICATION requires no special structures or restrictions. The BY VALUE and BY STRUCTURAL are both handled by the algorithm used to map the DBTG to the Universal. The algorithm uses the set definitions to place foreign keys (the set selection values) into the relational representation of the member of the set. Therefore any connection between the two must be through that set selection attribute.

DDL Mapping Issues

This section addresses issues of major importance which arose in developing the DDL mapping algorithms.

The Question of Third Normal Form. In two of the following mappings, network-relational and heirarchical-relational, the question arises as to whether or not the relational/universal view of the nonrelational data bases should be in 3NF. One might initially think that these mappings should be required to be in at least third normal form (3NF - see Chapter 1). However, it turns out to be that, at best, first normal form (1NF) can be ensured. Not only is 1NF the only form that can be ensured, it may be the only one desired. Essentially, network and heirarchical models are not based on the concepts of normal forms and, therefore, don't enforce them. First and foremost, the DBTG and IMS systems have

built in existence constraints (see Chapter 4, ER Constraints). The DBTG model runs into this problem through the different retention classes that it uses. A better example, though, is illustrated by the IMS model. In the IMS version of the medical data base (Appendix F), the segment LAB is a child segment of the the segment HOSPITAL. Because of the IMS hierarchical structure, a Lab cannot exist unless it currently serves at least one hospital in the data base. However, in the relational version of the medical data base (Appendix D), this is not the case. The LAB relation is independent of the HOSPITAL relation. This version, of course, obeys the restrictions of 3NF. The real impact of this difference comes in terms of updates and deletions. If a user attempted to delete a particular hospital from the relational data base, it would be relatively straightforward. However, this is not so the in the IMS version, since deleting a hospital segment would also delete all information about labs (among other things) that served that hospital. Another example would be attempting to add a new lab tuple. Via the universal model it would once again be a simple operation. However, if the actual underlying data base were IMS, this operation would cause an error because in the IMS data base the lab must be assigned to at least one IMS occurrence. This example leads to the second major difference between the universal/relational model and the network and hierarchical models. The latter two models/systems utilize a

great deal of replicated data (i. e. the lab information, assuming a given lab serviced more than one hospital, would be duplicated several times over). This fact is not evident in a relational view of the data. The main thrust or point of this discussion is that by violating 3NF in the universal model, it will assist the user and the system. Examine the universal, 1NF, relation below which is derived from the IMS version of the medical data base.

LAB

hospital code	lab code	Name	Address	Phone#

With this universal, 1NF version of the LAB relation, the user will realize that any new lab must first be assigned to a hospital to be inserted. A NULL value cannot be inserted because hospital code is specified as a NONULL value.

The end result of these issues is that the universal model will not hide as much of the underlying system as a normal relational system would. It should be pointed out that with this policy the DML mappings should be made easier and should reduce insertion and deletion anomalies that could arise since the relational queries will indirectly take into account more of the underlying data bases. It is important to note that while only 1NF is guaranteed, 3NF will be violated only when necessary as dictated by the underlying data bases.

LDBMS Modification vs Using Existing Data Bases. In many of the issues discussed in the DML and DDL mapping

sections, the question or option came up between either modifying the LDBMS to solve a particular problem or working within the existing data bases. The choice between these two was primarily one of choosing between the system and the users. As previously stated (chapter 2), the UDB is to stress the user's view over the system's view. For this reason and the fact that requiring the modification of the LDBMS could cause a great deal of trouble for the users, modification of LDBMS is not suggested as the best alternative. However, this is not an all encompassing policy and each such situation should be evaluated for the impact of any modifications versus the impact of not modifying the data bases.

The Question of Nonunique Keys. It is an unfortunate property that both the IMS and DBTG systems allow duplicate keys to exist. It is unclear how to handle this problem, therefore due to time constraints, this thesis will assume that duplicate keys are not allowed in the LDBMSs.

Keys and the UDB. In the algorithms presented in this chapter, and in general, it will be important that the UDB know what attributes in the underlying entities (relations, segments, or records) form the primary key for those entities. For this reason, in the Local Data Definition Language (LDDL) schemas this information will be available to the UDB.

DBTG Membership Classes. In the DBTG model, each member subentry involved in a set type must include a specification of the membership class for that set type. The membership

class affects the maintenance of the set in question (i. e. create, delete, or modify operations). The type of membership that exists is determined by two factors: the retention class and the insertion class.

The retention class is identified by one of three types: Fixed, Mandatory, or Optional. For discussion purposes, let us consider a set, OM, consisting of owner O and member M. A Fixed retention class indicates that once an occurrence of M has been entered into an occurrence of OM, it can never exist anywhere else in the set except as a member of that occurrence of OM (an occurrence can't change owners) (3:414). In relational terms, this would mean that, for example, any Ward code, wc, matched with a Hospital code, hc, could only be matched with that particular hc and no other. Another example involves a student data base with a TEACHER (Teacher#, other information) relation, a STUDENT (Student#, other information) relation, and a ADVISOR (Student#, Teacher#) relation. A fixed retention would mean once a student was assigned an academic advisor, he/she would have to keep that same advisor as long as that student was contained within the data base. The relational model, of course, allows for no such restriction. The user or the user's interfacing software is responsible for enforcing that articular kind of constraint. A Mandatory retention class indicates that once an occurrence of M has been entered into an occurrence of OM, it can only exist in the data base as

some occurrence of OM (3:413). In the medical relational example, this would indicate that Ward code could not be paired with any other key than the Hospital code. Consider the student data base described earlier, a mandatory retention would indicate that a student may change advisors throughout the period that a student's records are within the data base, but each student must always have an academic advisor assigned to him. Once again, the relational model (relational algebra) does not intrinsically provide this constraint. It could be accomplished by providing additional integrity constraints such as those described by Date (3). The final retention class, Optional, places no restrictions upon the set. The relational model would be considered to have an Optional retention class.

There are two classes of Insertion: Automatic and Manual. The Automatic insertion class indicates that when an occurrence of M is entered into the data base, the DBMS will connect it into the data base in the appropriate occurrence of OM (program must generally specify which occurrence of OM) (3:414). In the medical data base, an example would be that if a test were ordered (thus inserted in the TEST record), it would automatically insert an entry into the TEST ASSIGNED and TESTS ORDERED sets. The Manual insertion class indicates that the user's application must explicitly issue the necessary connect command to insert any occurrence of M into OM (3:414).

The next question is, of course, how are the membership classes going to be represented in the universal/relational model and/or mapped into the DBTG network model?

The retention classes can be handled in a relatively straightforward manner. A Fixed class retention is indicated in the DDL by a "Unique associations" clause (see Chapter 7). It is important to note that this only notifies the user that he/she should make these associations unique; it is up to the user to police this outside constraint. Obviously, this is not the most eloquent solution but one that, in theory, should work. It is a simple solution and is, perhaps, better than not allowing Fixed retention at all. A Mandatory retention class is a default class due to the DBTG to Universal algorithm which automatically places all relations/sets into a mandatory retention class. An optional retention class requires that the key of the owning record is permitted to be NULL in the member record. An example would be in the STAFF DOCTORS set which has HOSPITAL as the owner and DOCTOR as the member set. Therefore, the DOCTOR record would allow its Hospital code to take on a NULL value (see Chapter 4, page 77, Integrity Rule #2).

The insertion classes prove a bit more difficult to solve. Manual insertion requires no modifications or algorithm to accomplish. A brief examination of possible solutions to handling Automatic insertion shows it to be complex and probable solution(s) to be very cumbersome

(particularly when one considers cases of different models represented by the same universal relation). Therefore, in this thesis effort, the Automatic retention class will not be allowed.

Distributed Information. The issue of distributed information in the DDL mapping context is a relatively straightforward problem. The algorithms presented in this chapter map a local heirarchical, relational, or network data base into a universal/relational representation. However, these individual representations must still, if desired, be integrated with other data bases (or portions of other data bases). Therefore, after the required universal data base representations are generated, they are to be integrated together and evaluated. The evaluation, done by the UDBAC, involves removing redundant, horizontally split, and vertically split relations.

Redundancy involves removing redundant relations. The removal of a redundant relation or data base requires that they be completely identical or at least, if having alternate names or other constraints, combined in a manner which is lossless.

A horizontal split is a case where a relation is comprised of attributes which are physically located at different sites. Such relations add complexity but may not be avoidable. A horizontal split may also occur when two or more data bases are storing very similiar data (such as the

three medical data bases used in this thesis). An example would be where a relational data base in the network has a universal (as well as relational) relation with three attributes A, B, and C. One, or perhaps two, other data base contains a generated universal relation with attributes A and B; and another with A and C. Obviously, in a normal relational system, the correct relation would have A, B, and C in one relation. In the UDB, placing them together would cause some additional complexity in the DML queries but not a significant amount. This thesis effort will assume that such situations are resolved as if they were in a normal DBMS. Another possible reason for such a situation to exist, other than the physical one already discussed, would be where a certain number of data bases might have a relation with attributes A, B, C, and D, while others might only have A, B, and C. There are four possible solutions to this situation. The first is to have separate relations. Unfortunately, the user must now find all of the pertinent relations in a data base. The obvious problem occurs when the user is performing an update, misses a relation, and now the data base has an integrity violation. The second is to remove attribute D from those data bases having it; restructuring those data bases to retain the information contained in D or the third is to add attribute D to those who did not already have it, possibly restructuring those data bases. This solution is the easiest from the system's viewpoint but could be costly and difficult

for the users to accomplish. A fourth solution is to make the universal representation appear to have the extra attribute (not a key) when not all of the underlying data bases do. This is certainly easier for the user but adds some complications to DML translations. If the added attribute does not exist anywhere else in the underlying data base(s), then it will have to be treated as a NULL allowable attribute. If the attribute does exist else where in the data base, then any retrievals, updates, etc., will have to correctly process that attribute in some manner. It is important to note that while this solution appears to be the best solution of the four (and will be assumed for this thesis), it is not clear that the DML translations will always be able to correctly handle such situations. The final decision on this solution will have to wait until the DML translations are examined more closely.

A vertical split is a case where two or more relations are identically structured but contain different information (medical data bases good example). These relations will be represented as a single relation.

Redundant Data. While the issue of distributed information presents some difficulties, the issue of redundant data proves even more so. Before discussing the problem it is important to understand exactly what is meant by redundant data in the UDB context. Redundant data is in general is any information which is repeated verbatim in more than one place

(DBMS). The solutions to the problems of redundant data depend on the policy that is established. Will the UDB allow redundant data to exist? Would it be better to only allow completely redundant data? Or finally, must the UDB support partially redundant data (as well as fully redundant data)? In this subsection, the above issue will be addressed first with each described in terms of its advantages and disadvantages. Then the two possible cases of redundant data will be discussed in terms of each of the three possible redundant data policies. The section will conclude with a recommendation as to which approach to use.

The first possible policy toward redundant data would be not to allow it to occur. At first one might think this policy without any benefits. This is not true. First, such a policy would eliminate any problems with redundant data retrievals, updates, and deletions, greatly reducing the complexity of mapping these commands over the network. Second, since the environment is stated as being a cooperative one, the UDBAC will have the power to move information around in a any manner it desires to improve the efficiency of the system. The users would never know that the information had been moved (Note: this does have a ramification for the LDBMSs from where the data was removed. This could be solved by having all queries done in the UDM or having the LDBMS make a system-generated request to the network for the information that had formerly been at that LDBMS). Removing

redundant information (except for backups) rom the system would free up a great deal of memory for other uses. Unfortunately, there are also problems with this policy. First, this will greatly increase (depending on the data) the activity on the network, both in terms of queries and data transferal. This will also slow down the system's response time. Finally, if that particular site goes down, that information would be lost to the network until it was brought back up or the backup copy was activated (Note: Using backups, in a certain sense, makes this policy more like the fully redundant data policy depending on how the backup's integrity is maintained).

The second possible policy is the fully redundant data policy. The first advantage of this policy is that extra retrieval processing would be eliminated since only one of the N number of sites would have to be queried. Secondly, although all sites would have to updates and deletes sent to them, the queries (UDML) would all be identical. Thirdly, the UDBMS could route a query (retrieval) to one copy of the data over the others depending on the workload at the different LDBMS involved. And finally, with multiple copies there is automatic backup in case a system goes down. If one copy goes down for a certain period, then when it comes back up it could be backed up one of two ways. First, a list of any modifications could be maintained by the UDBAC and executed against the LDBMS when it comes back up, or a current copy

from an unaffected site could be transferred in mass to the formerly down site, replacing the old information. The disadvantages to this policy parallel its strengths. Multiple copies require N times the work to update and delete assuming N copies present in the network. Multiple copies take up much more memory in general and, depending on the local user's needs, may involve storing a great deal of unnecessary information (for that LDBMS). And if the information is not stored there, then the amount of network activity and the transaction response time will both go up. The real blow to this policy is the amount of memory that will be required to support it. It is possible that under certain circumstances, this policy could be used but not as a data base wide one. It is important to note that both of the first two possible policies involve local restructuring of the data (see discussion in this chapter on this subject).

The third, and last, possible policy is to allow partially redundant data, i. e. basically allowing the systems to exist as they are. The primary advantage of this policy is that it requires no LDBMS modifications and places no restrictions (at least a minimal set) upon the local users. From the local viewpoint this could reduce the network activity and increase transaction response time because the information most often needed could be stored locally. This policy certainly has some disadvantages. Allowing partially redundant data will increase the complexity of integration of

the individual universal representations of the local data bases. With this policy, retrievals will require extra processing since redundant data will be examined several times and it will require a natural join at the UDB level to eliminate the redundancy (see redundant data discussion in the DML mapping issues section). This will require extra data to be sent along the network (along with the extra queries). Updates and deletions pose similiar problems along with some additional integrity problems. While all of these disadvantages are not desired, the end result is that partial edundancy is unavoidable. One might avoid it in terms of "redundant structures" but not in terms of redundant attributes. Consider how many times a person's name is repeated in a UDB linking all of the base personnel DBMSs in the Air Force and the main personnel data base at Manpower and Personnel Center in Texas. It seems extremely undesirable to have all of that information stored at one place (at least with communication costs today) or to have all of the sites have all of the personnel data. Furthermore, it is decidedly inconvient to allow a person's name to only appear in one place. Clearly, partial redundancy must be handled, whatever the cost. Having discussed the three different possible policies towards redundant data, let us examine the two possible types of redundant data that could occur.

The first type of redundant data involves two or more relations which are identically structured (in the universal

representation) and contain the exact same information or portions of the exact same information. There are several options in handling this situation. If it is known that the information is fully duplicated (and will remain that way) then the the two (or more) relations would be combined (in the universal view) and appear as one relation. It would be indicated in the data dictionary that the two different physical underlying "relations" were composed of fully duplicated data. If full duplication can't be guaranteed or is not intended at all, then the different relations can be represented as separate relations or as one relation but with added complexity and processing.

The second type of redundancy involves two (or more) relations which are identical except for some extra information in one (or more) of the relations. An example illustrates:

PRESENT PATIENTS	PAST PATIENTS
A B C	A B C D

There are several possible solutions to this problem depending, on which redundancy policy is in effect. Perhaps one relation is for PRESENT PATIENTS and the other for PAST PATIENTS and it is desired that they be separate. In the case were one relation merely has an additional attribute or that attribute is somewhere else in the data base's individual representation, then modifications might have to be made or the additional complexity and burden placed upon the sys-

tem with the DBA making these decisions. The possible modifications include physically adding, deleting, allowing the attributes to be NULL, and even creating another relation (at the UDB level). The reader is referred to the section on UDDL Integration at the end of this chapter.

Relational Constraints

This section summarizes the limitations that are being placed on the System R relational model/system.

1. The System R schema will inform the UDB as to which attribute(s) in a relation is the primary key.

IMS Constraints

This section summarizes the limitations that are being placed on the IMS heirarchical model/system.

1. Duplicate keys are not allowed.
2. Unless otherwise noted, the abilities and powers of the IMS system are as presented by Date (3).
3. The IMS schemas will inform the UDB of the primary key(s) in each IMS segment.

DBTG Constraints

This section summarizes the limitations that are being placed on the DBTG network model/system.

1. Duplicate keys are not allowed.
2. Only optional and mandatory retention classes are "policed" by the system. Fixed retention is user "policed".
3. Only manual insertion is allowed. Automatic insertion is not.

4. Unless otherwise noted, the abilities and powers of the DBTG network system are as per Date (3).

5. The DBTG schema will inform the UDB as to which attribute(s) in each DBTG record is the primary key.

Data Definition Language Mappings

The purpose of this section is to examine the mappings between the UDM and relational model, the UDM and network model, and the UDM and the heirarchical model in terms of their respective data definition languages. Each discussion will describe a general approach or algorithm to performing the respective mapping. An algorithm is described for the LDDL to the UDDL and the UDDL to the LDDL. The former is what is used by the UDBAC to generate a universal representation of each physical data base. The latter is merely used to show that the original underlying data base can be recaptured. In actual practice, the Data Dictionary will contain the necessary information to accomplish the UDDL to LDDL mapping if and when required. This section merely describes how the individual representations are generated. It does discuss how these individual representations are integrated into a single UDB. The reader is referred to the Integration section at the end of this chapter.

The approach taken in the DBTG mappings algorithm is initially based on Larson's (7) work. The IMS algorithm is an extension of this due to the fact that the heirarchical model is a subset of the network model. All examples for the algorithms presented in this section are from the sample data

base presented in the appendices. The reader is referred to Appendices C-G.

Mapping Algorithm Key. The following definitions and abbreviations for attributes are used in explaining the algorithm to follow.

Primary key: An attribute or set of attributes within a tuple, segment, or record which uniquely identifies that tuple, segment, or record. Example: In HOSPITAL, Hospital code is is the primary key.

Foreign key: An attribute within a tuple, segment, or record which is a primary key in another tuple, segment, or record. In WARD, Ward code is the primary key and Hospital code is a foreign key.

Hospital code = HC Ward code = WC Doctor# = DOC#
Diagnosis code = DC Employee# = EMPL# Test code = TC
Registration# = REG#

Notation: Names in all capital will denote the name of a relation, segment, record/set or one of the abbreviations for attributes outlined above. It will also be used to convey the original contents of that relation, segment, or record/set (i. e. In the context of the DBTG record named HOSPITAL, the mapped relation HOSPITAL might only be listed like below:

```
============
= HOSPITAL =
============
```

But would actually be conveying:

HOSPITAL

```
=========================================
= HC = Name = Address = Phone# = # of beds =
=========================================
```

If during a mapping algorithm, an attribute, WC, was added to a relation, WARD, it would be shown as follows:

```
=============
= WARD = WC =
=============
```

It would actually be conveying the following:

WARD

WC	HC	Name	# of beds

Universal-Relational Mappings. These two mappings, universal DDL to relational DDL and vice versa, are both trivial since the universal model is also a relational model. Any mappings would only involve syntax translation. Therefore, these two mappings will not be discussed.

Network DDL to Universal DDL. This mapping appears to be a relatively straightforward algorithm. The network (DBTG) model has two types of structures, records and sets. Records and sets more or less correspond directly to rela-

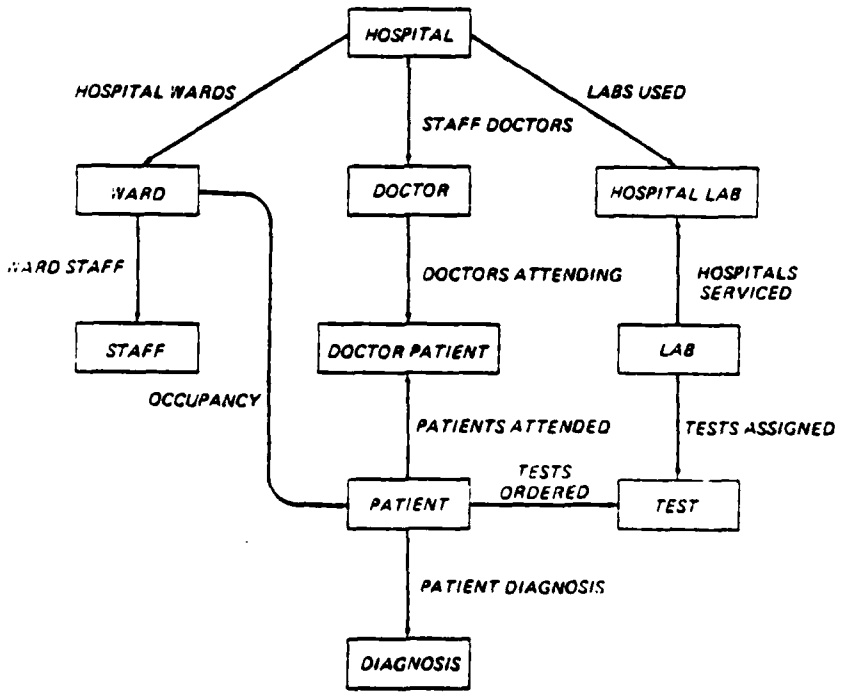


Figure 41. DBTG Version of the Medical Data Base (9:121)

tional tuples although additional attributes may be required to make the record a true tuple. These additional attributes are determined by the owners and the corresponding set selection values of any sets a given record is involved in. The mapping algorithm is as follows:

1. Place the attributes of each record (RECORD NAME IS) into a relational tuple format.

   Specific example:

   STAFF

Empl#	Name	Duty	Shift	Salary

   Mapping status after step 1:

HOSPITAL	WARD	STAFF

DOCTOR	DOC-PAT	PATIENT

DIAGNOS.	TEST	LAB

HOSP-LAB

2. Any sets which are structurally set selected (SET SELECTION IS BY STRUCTURAL) are formed into relations with the relation consisting of the primary keys of the two records involved. These sets are removed from consideration.

   Specific Examples:

   DOCTORS ATTENDING

DOC#	REG#

   PATIENTS ATTENDED

REG#	DOC#

LABS-USED	HOSPITAL-SERVICED
= HC = HC = Lab# =	= Lab# = HC = Lab# =

3. Examine all sets in which a record, now relation, participates in as a member (MEMBER IS) and the retention class of that set is OPTIONAL. These sets are formed into relations in the same manner as they were formed in step #2. These sets are removed from further consideration.

   Specific example:

   STAFF DOCTORS

   = HC = DOC# =

   Mapping status after step 3:

   = HOSPITAL = = WARD = = STAFF =

   = DOCTOR = = DOC-PAT = = PATIENT =

   = DIAGNOS. = = TEST = = LAB =

   = HOSP-LAB =

   DOCTORS ATTENDING

   = DOC# = REG# =

   PATIENTS ATTENDED

   = REG# = DOC# =

   LABS-USED

   = HC = HC = Lab# =

   HOSPITAL-SERVICED

   = Lab# = HC = Lab# =

   OCCUPANCY

   = WC = REG# =

   STAFF DOCTORS

   = HC = DOC# =

4. Examine all sets in which a record, now relation, participates as a member. Add the attribute which functions as the set selection value (SET SELECTION IS BY VALUE OF) to the member record/relation. Added attributes are considered foreign keys within the relation. If the retention class is fixed, then a unique association between the key passed and the primary key of the receiving relation is indicated (in the UDDL).

Specific example:

STAFF

Empl#	WC	Name	Duty	Shift	Salary

Mapping status after step 4:

HOSPITAL

WARD	HC

STAFF	WC

DOCTOR

DOC-PAT

PATIENT

DIAGNOS.	REG#

TEST	Lab#	REG#

HOSP-LAB

LAB

DOCTORS ATTENDING

DOC#	REG#

PATIENTS ATTENDED

REG#	DOC#

LABS-USED

HC	HC	Lab#

HOSPITAL-SERVICED

Lab#	HC	Lab#

OCCUPANCY

WC	REG#

STAFF DOCTORS

HC	DOC#

5. Examine all relations in which attributes were added in in step #4. If there are three relations (a, b, c) and in step 4 an attribute, X, was passed from a to b and

an attribute, Y, from b to c, then step #5 would pass X to c. In the example below, HC was passed to WARD and WC passed to STAFF in step #4. Step #5 now passes HC to STAFF.

Specific example:

STAFF

Empl#	WC	HC	Name	Duty	Shift	Salary

Mapping status after step 5:

HOSPITAL

WARD	HC

DOC-PAT

DOCTOR

STAFF	WC	HC

PATIENT

DIAGNOS.	Reg#

TEST	Lab#	Reg#

HOSP-LAB

LAB

DOCTORS ATTENDING

DOC#	REG#

PATIENTS ATTENDED

REG#	DOC#

LABS-USED

HC	HC	Lab#

HOSPITAL-SERVICED

Lab#	HC	Lab#

OCCUPANCY

WC	REG#

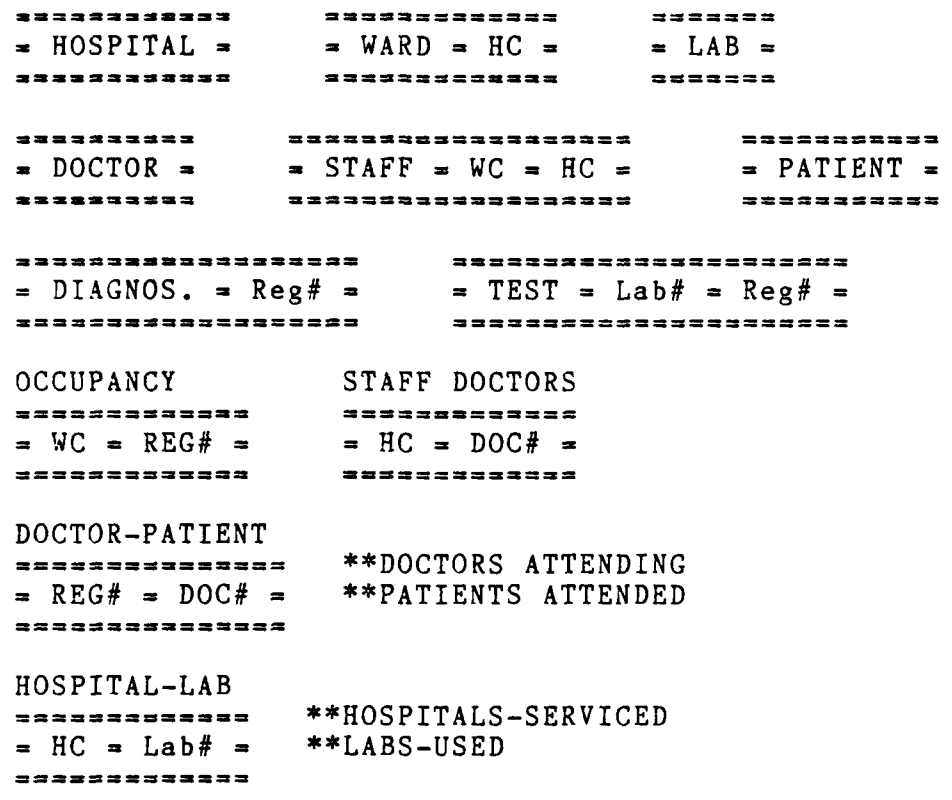
STAFF DOCTORS

HC	DOC#

6. Remove any redundant attributes within any relations.

7. Remove any redundant relations. These relations should be marked as alternate names relations.

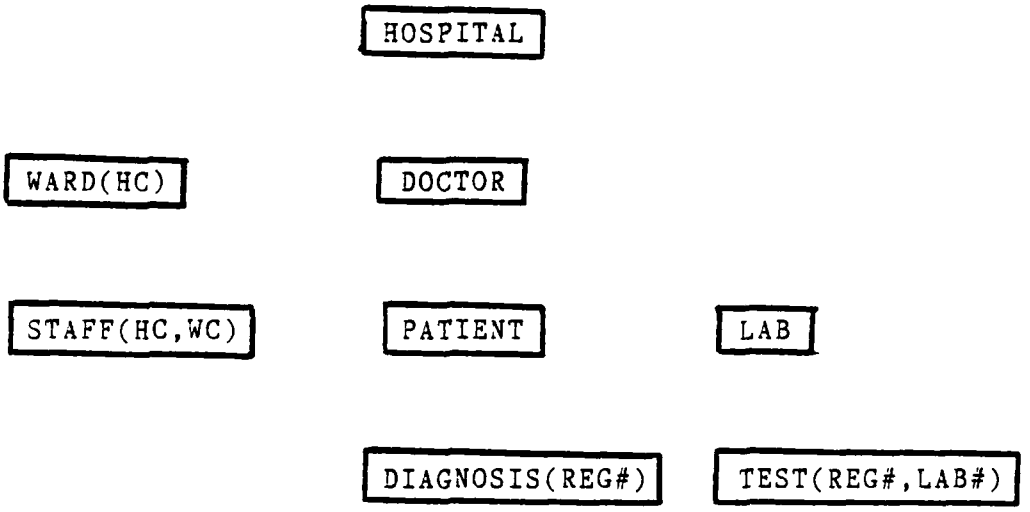
Final mapping after steps 1-7:



Universal DDL to Network DDL Mapping. This mapping appears to be fairly straightforward. For the most part, relations can be directly mapped into DBTG records after removing any foreign keys. The foreign keys indicate where DBTG sets should appear. Relations composed of only foreign keys may also indicate where DBTG sets should appear. The UDDL to Network DDL algorithm is as follows:

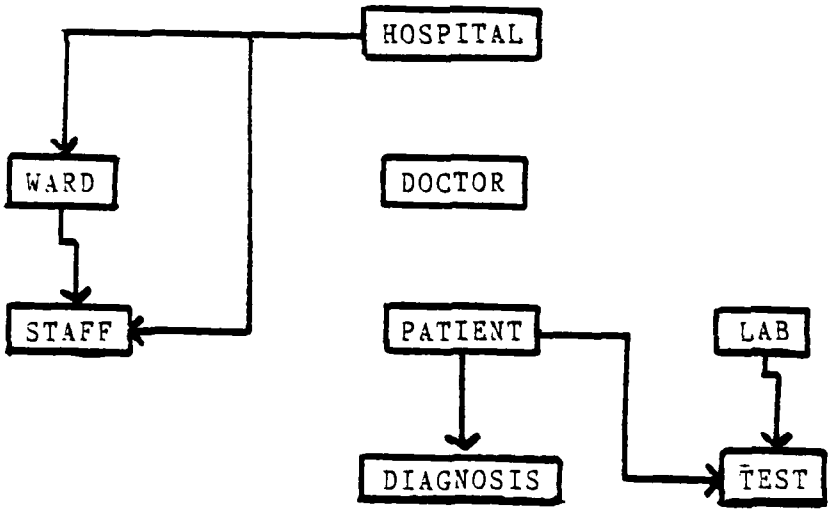
1. Start with relations which are not composed entirely of foreign keys. Form them into records, removing the foreign keys but noting which foreign keys were formerly associated with that newly formed record.

Mapping status after step 1:



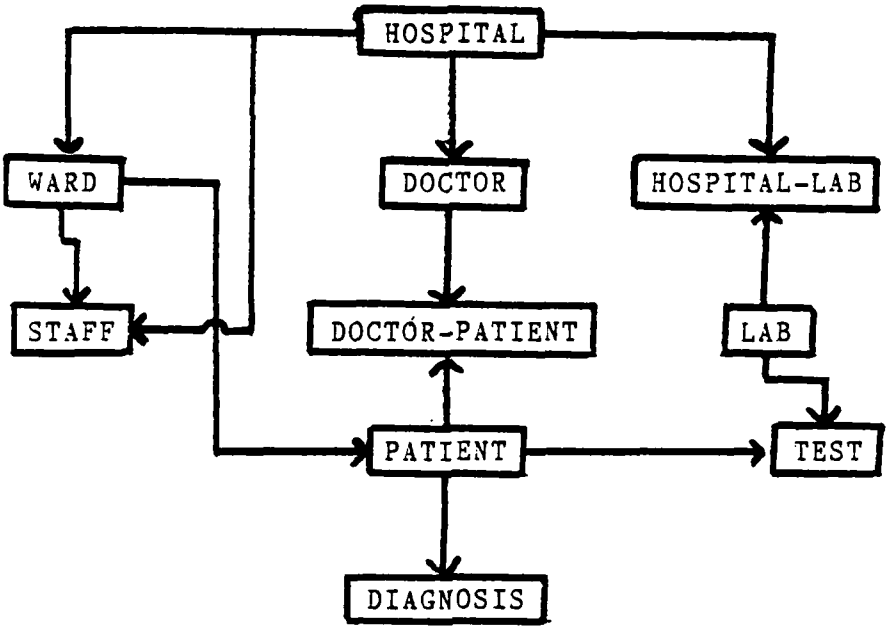
2. Determine where each foreign key is the primary key and connect that record with the record which formerly had the foreign key as part of the original relation. The owning record is the record for which the key is primary.

Mapping status after step 2:



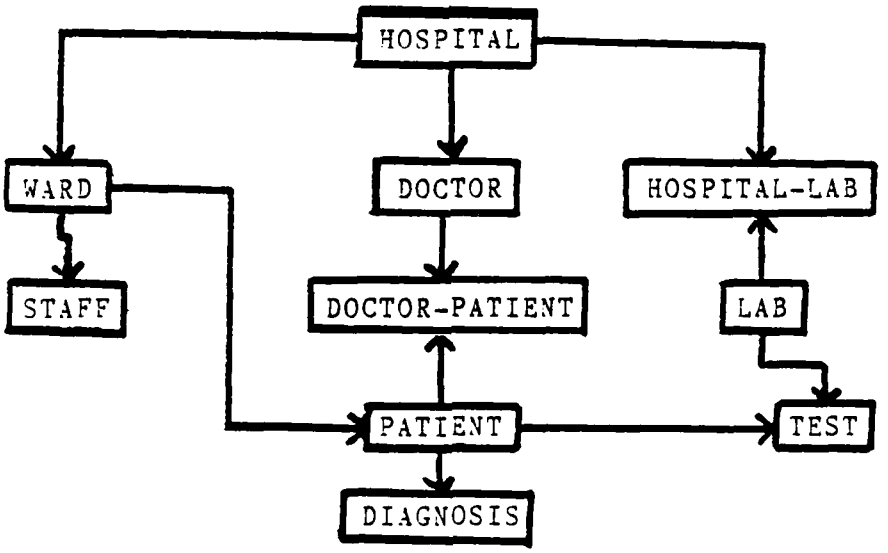
3. Consider any relations consisting of only two attributes, both foreign keys. If the relations have alternate names, then the relations are formed into records with the foreign keys as attributes. These records are connected via structural set selection sets. Which records are to be connected is determined by the for-

eign keys. The records for which these foreign keys are the primary keys are connected. If the relations do not have alternate names, then these relations specify which records are to be connected (the owner/member is specified in the data dictionary).



4. Examine all paths between records. If there is more than one way to get from a given record A and another record B, eliminate last link in all but the longest path.

Mapping status after step 4:



By comparison to Appendix E, one can see that the original DBTG data base mapped in the previous section has been recaptured.

IMS DDL to Universal DDL. This subsection presents two approaches to mapping the IMS DDL to the Universal (relational) DDL. The first approach, the Parent-Key algorithm, forms relations by pushing the key of the owner sets into the member sets as foreign keys. This results in a universal (relational) representation which is not in 3NF (only 1NF) but captures the restrictions for entering, updating, and deleting data within the underlying IMS data base. The second approach, the Link algorithm, forms relations by making each record and set distinct relations. The resulting universal (relational) data base is in 3NF but has a larger number of relations than are necessary.

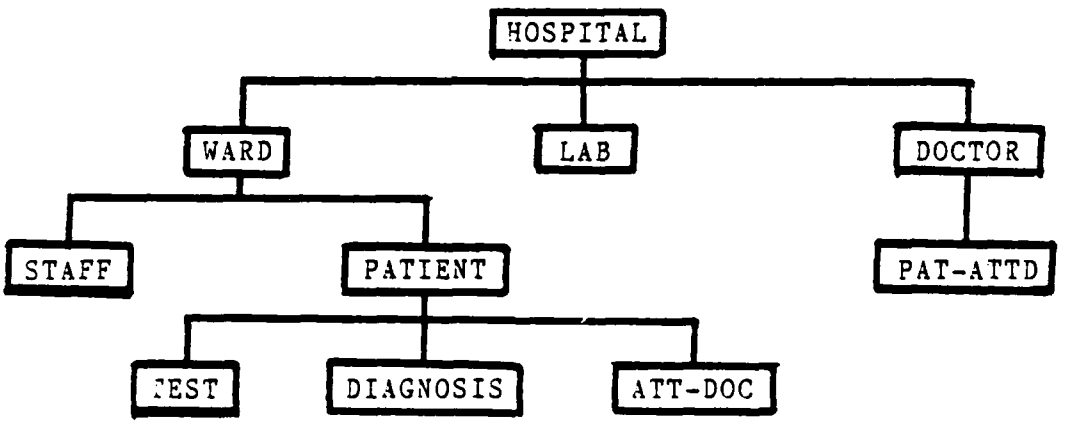


Figure 42. IMS Version of Medical Data Base (9:149)

Parent Key Algorithm.

1. Convert all IMS segments to a relational format.

Specific example:

HOSPITAL

Hospcode	name	address	phone#	#ofbeds

Mapping status after step 1:

HOSPITAL	WARD	LAB	DOCTOR

STAFF	PATIENT	PAT-ATTD	TEST

DIAGNOS	ATT-DOC

2. All children segments/relations add the primary key of their parents.

Specific example:

WARD

Wardcode	Hospcode	name	#ofbeds

Mapping status after step 2:

HOSPITAL	WARD + HC	LAB + HC

STAFF + WC	PATIENT + WC

DIAGNOS + REG#	ATT-DOC + REG#

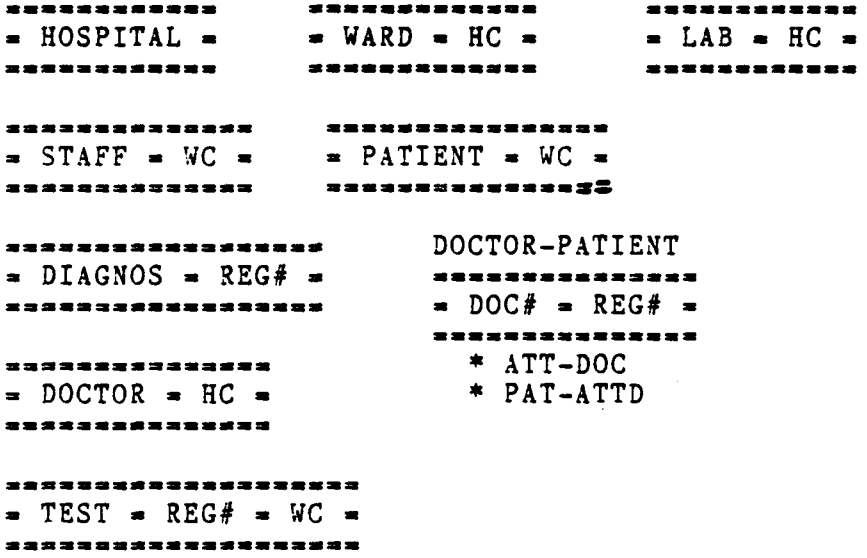
DOCTOR + HC	PAT-ATTD + DOC#

TEST + LAB# + WC

3. Remove any redundant attributes within a relation.

4. Remove any redundant relations within the data base. These relations are marked as alternate names.

Mapping status after step 4:



5. Examine all relations/segments according to the IMS sequence number. If any relation contains added attributes which are paired in a relation/segment with a lower sequence number, then remove the attribute which originated from the higher sequence number.

   This step removes the attribute WC from TEST. The final resulting mapping is as in step 4 but with WC removed from TEST.

Link Mapping Algorithm.

1. Convert all IMS segments to relational format.

   See step #1 in Parent Key algorithm for mapping status after this step.

2. Convert all Parent-Child relationships to relational format.

Mapping status after step 2:

LAB#	HC

DOC#	REG#

TC	REG#

HC	WC

EMPL#	WC

DC	REG#

DOC#	HC

REG#	WC

REG#	DOC#

3. Remove redundant relations and mark remaining representitive as alternate names.

Mapping status after step 3:

* Alternate Name

LAB#	HC

DOC#	REG#

TC	REG#

HC	WC

EMPL#	WC

DC	REG#

DOC#	HC

REG#	WC

4. Remove any relations consisting of only one attribute.

This removes ATT-DOC and PAT-ATTD.

Final result of Link Algorithm:

HOSPITAL

WARD

LAB

DOCTOR

STAFF

PATIENT

DIAGNOS

TEST

LAB#	HC

DOC#	REG#

TC	REG#

HC	WC

EMPL#	WC

DC	REG#

DOC#	HC

REG#	WC

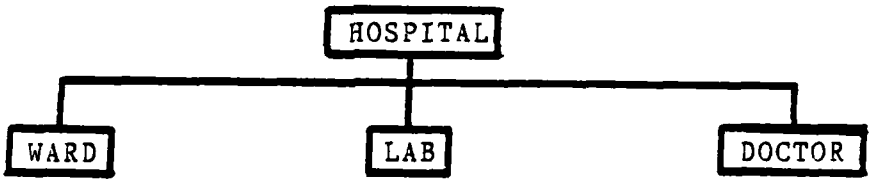
Universal DDL to IMS (Parent-Key) DDL. This section describes the algorithm to recapture the IMS DDL from the Universal DDL representation of it generated by the Parent-Key algorithm.

1. Examine all relations and find the relation with no foreign keys. That relation is the root segment.

   Mapping status after step 1:

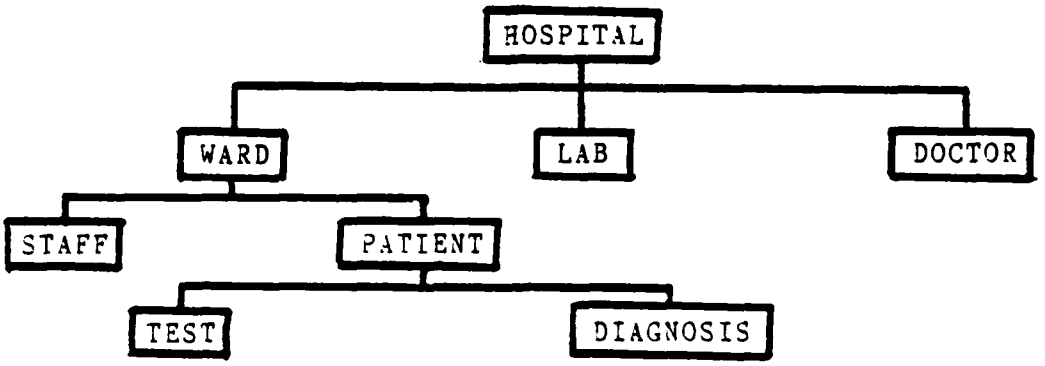
2. Examine each relation which has the primary key of the root (HOSPITAL) as a foreign key. These rel- become the children segments of the root. Only consider relations with one foreign key.

   Mapping status after step 2:



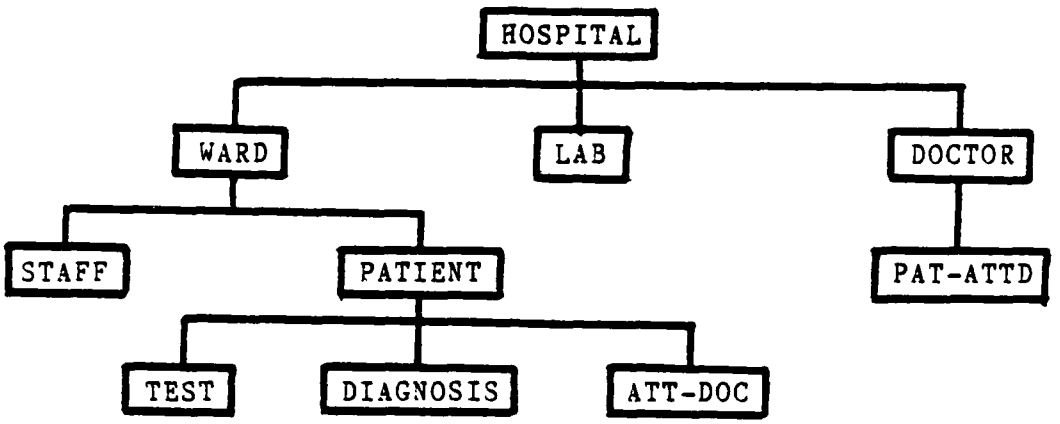
3. Repeat step #2 recursively on the children segments. Only consider relations with one foreign key.

   Mapping status after step 3:



4. Now examine all alternate names relations. Designate each foreign key as f1, f2, etc., and any other attributes collectively as A. Take each foreign key determine what segment, S, has that key as the primary key. Form the other foreign keys and A into a segment and make that new segment a child segment of S. Repeat this for each foreign key.

Final mapping status:



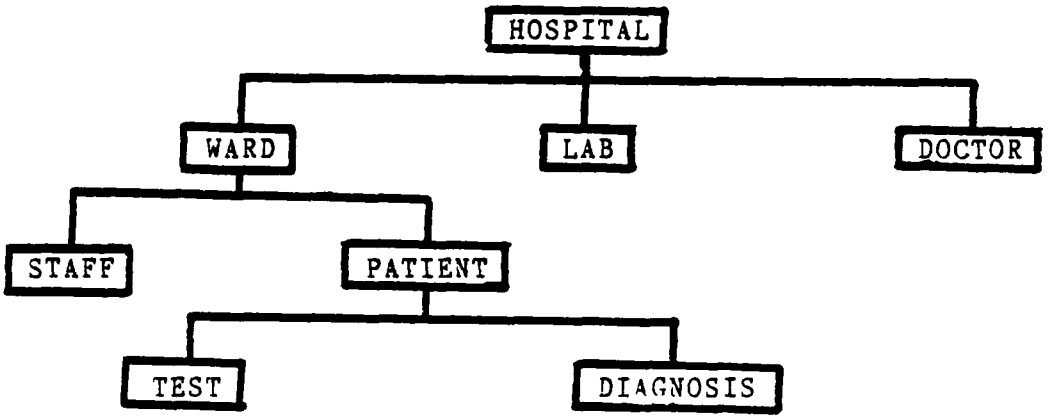
Comparison of the above IMS structure and the one used as input to the Parent-Key algorithm proves that the original IMS structure can be recaptured.

Universal DDL to IMS (Link) DDL. This section shows that the universal representation of the IMS data base generated by the Link algorithm can be recaptured (i. e. a two mapping exists).

1. Form all relations consisting of keys and attributes into IMS segments. Do not consider alternate names relations.

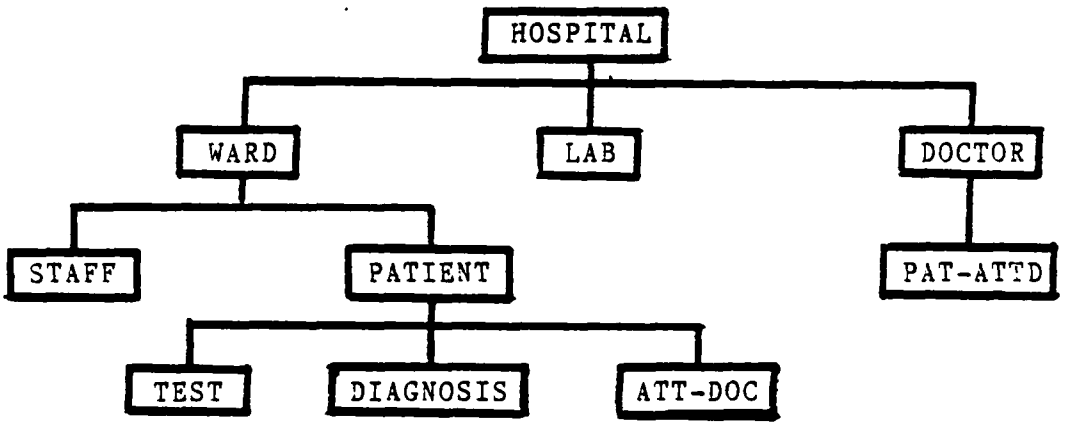
2. Using the IMS sequence numbers contained within the data dictionary and the relations consisting of only two keys to connect the segments created in step #1 into the correct heirarchical structure. This is done by examine a relation consisting of two keys and determine what segments have those keys as primary keys. Whichever segment has a lower sequence number is the parent segment, the other the child segment. Do not consider alternate names relations.

Mapping status after step 2:



3. Now examine all alternate names relations. Designate each foreign key as f1, f2, etc., and any other attributes collectively as A. Take each foreign key determine what segment, S, has that key as the primary key. Form the other foreign keys and A into a segment and make that new segment a child segment of S. Repeat this for each foreign key.

Mapping status after step 3:



Comparison of the above IMS structure and the one used as input to the Link algorithm proves that the original IMS structure can be recaptured.

This subsection has presented two algorithms to this particular mapping. The Link mapping algorithm is simpler

and in 3NF. The Parent-Key is more complicated, is only in 1NF, but does capture more explicitly some of the underlying data constraints. By doing so, the Parent-Key assists the user and system in handling system updates, insertions, and deletions. Therefore, this thesis chooses the Parent-Key algorithm as superior.

Integration of UDDL Mappings

The concept of the integration of the various individual UDDL mappings takes those relatively (?) simple mappings and complicates them. It is important to note several points about the integration process:

1. It will be a manual process performed by the UDBAC.
2. It will require an understanding of the underlying physical data bases.
3. It will require a good understanding of the purpose or requirements of the universal data base to be created from the UDDL mappings.

UDB Relations. It may occur in the UDB that when integrating several like data bases that there may not be intrinsic relations or attributes within the underlying data bases which will distinguish the information within them. For example, there is no direct attribute or relation in the medical data base example which indicates in what city a Hospital is located, or in what city a Patient lives, etc. There are several different solutions to this problem. The first is to modify the underlying data bases to distinguish these physical locations. This type of modification will most likely not be as disrupting as other types previously

mentioned. It should be noted that it may prove very desirable to perform this type of modification. This is because it will facilitate the movement of data bases from one system to another (i. e. the personnel information for Wright Patterson Air Force Base might be moved to Scott Air Force Base where all personnel information for the midwest is being moved). Another possible approach is to create a new relation at the UDB level, in the UDBAC computer, to reflect any location information about a data base. This is easier from the user's point of view but may cause more network traffic and, of course, must now be maintained. Then again the UDB could just be treated as another relational system on the network. Both of these solutions are viable ones, the exact one chosen depends on the requirements of the particular data base(s) in question.

UDDL Integration Methodology. There is no well defined algorithm or process for the integration process. Like the design of a data base, it is a creative process with too many possibilities to be automated. But as with data base design, computer-aided tools will greatly improve the efficiency of the process. Although there is no set procedure, a few general guidelines and examples can be established. It is important to note that some situations may arise which are not readily solved and may require such things as LDBMS restructuring and so forth.

Before going over the integration of the medical data

base used throughout this thesis, a few generic examples are outlined below. Examples deal with only two relations but they can be generalized to more than two.

1. Two relations, one with attributes A and B; and the other with attributes A and C.

   This particular example, discussed earlier, can be be solved by either placing all three attributes together (A B C) or leaving them separate as two relations.

2. Two relations, one with attributes A, B, C, D; and one with attributes A, B, C.

   Another example already discussed, this can be solved in one of two ways also. The first is to have one relation with A, B, C, and D. The second is to have one relation A, B, C and another (assuming that A is the key) A and D. The choice depends on the actual attributes involved.

3. A final example involves three relations, one with A, B, C; one with A, B; and a third with C, D, E.

   This one is solved with one relation with A, B, and C; and one relation with C, D, and E.

Integration Example. This section traces through the integration that took place to produce the UDB version of the medical data base in Appendix G. The example takes the individual mappings derived for each of the three different versions of the data base and attempts to integrate them. It is important to note that this particular example deals with data bases dealing with exactly the same type of information thus making the integration tougher. The reader will need to refer to Appendices D, E, F, and G.

1. The first thing to do is to integrate those relations which are identical. Alternate names are combined as well. Any modifications refer to the universal representations, not the actual under-

lying structures.

Effected relations:

HOSPITAL: integrated without problem.

WARD: integrated but unique association clause kept. A ward can't be with more than one hospital anyway so there are no consequences for System R or IMS DBMSs.

DOCTOR PATIENT: Combination of a normal relation from System R and two alternate name relations from DBTG and the IMS versions.

DIAGNOSIS: Integrated with no problem.

2. Now each remaining relations must be examined one at a time.

HOSPITAL-LAB: The System R and DBTG versions prove identical (aside from alternate names for DBTG version). Examining the IMS version, it is noted that the LAB relations are all identical except that the IMS version has Hospital code added. First impulse is to remove Hospital code from LAB along with Lab# to form the third HOSPITAL-LAB. However, Hospital code is needed in the IMS LAB to indicate that a lab must be assigned to a hospital before it can exist within the data base. Therefore, the reverse is done. HOSPITAL-LAB is removed, and Hospital code is added to the System R and DBTG versions of LAB (the alternate names from the DBTG version is retained as well).

LAB: see HOSPITAL-LAB.

STAFF: The System R and DBTG versions are identical and are combined. The IMS version is missing Hospital code. Examine the IMS physical structure (Appendix F), it is noted that Hospital code (for insertions) is required and would most likely be be used in any queries anyway. Therefore, Hospital code is added to the IMS version and it is combined also.

TEST: The System R and DBTG versions are identical and are integrated. The IMS version has an additional attribute, Ward code. Examining the IMS physical structure, it is noted that while the Ward code would be useful, most queries will be based on the

Registration# and/or Test code. Therefore, Ward code is deleted from the IMS version and it is combined with the other two versions.

DOCTOR: The IMS and System R versions are identical and are combined. The DBTG version lacks the attribute, Hospital code. Examining the DBTG version, it is noted that an extra relation, STAFF DOCTORS, provides the necessary connection. Therefore, STAFF DOCTORS is deleted and Hospital code added to the DBTG version and combined with the other two.

PATIENT: All three versions in this situation are unique. Examining the IMS and DBTG versions, both have bed# but the IMS version has Ward code also. It is further noted, that the relation OCCUPANCY (DBTG version provides the necessary connection. Therefore, these two are combined. Now considering the System R version, it is determined that placing (physically or logically) the attribute, bed#, into the System R version of PATIENT or not, is a matter of choice. Therefore, it is added to the System R version and removed from the System R version of OCCUPANCY. It is also noted that the OCCUPANCY information is present in all three versions, just in a different manner. Due to the need to portray underlying restrictions in the IMS and DBTG versions of the data base, we choose to implement the IMS/DBTG version. Therefore, Ward code is added to the System R version of PATIENT and OCCUPANCY is deleted.

HOSPITAL LOCATION: This relation is created by the UDBAC to distinguish the information between the hospitals in the different data bases. It is composed of the Hospital code and a Location code.

LAB LOCATION: This relation is created by the UDBAC to distinguish the information about the labs in the different data bases. It is composed of the Lab# and a Location code.

PATIENT LOCATION: This relation is created by the UDBAC to distinguish the information about the patients in the different data bases. It is composed of the Registration# and a Location code.

Conclusion

In this chapter, many of the more complex and fascinating issues of the UDB have been briefly addressed. The complete and successful resolution of the issues and problems raised in this chapter will be a must for the actual implementation of the UDB.

Chapter VII. The Universal Data Definition Language, the Universal Data Manipulation Language, and the Data Dictionary

Overview

The purpose of this chapter is to describe the syntax of the UDDL and UDML, and the composition of the Data Dictionary. The UDDL is based on System R as described in Date(3), while the UDML is based on Quest, a language designed and developed by Housel (5). Both the UDDL and UDML are described in a BNF format with the UDDL being described first. Comments are inserted at certain points for clarafication. They are identified by parenthesis and asterisks, (* comments *). **Bold** indicates an entity within the system, not a part of the BNF syntax. The Composition of the Data Dictionary is described in a relational manner. The purpose is to show what information the Data Dictionary must contain in order to perform DDL mappings and DML translations.

The Universal Data Definition Language

The Universal Data Definition Language specifies the relational structures that the distributed users will see as the UDB. The approach taken in designing this language was to first design a powerful DDL for a standard relational system and then modify it to handle the special cases and problems that would arise in the UDB environment. The language is described in two parts, the general definitions

and the command definitions.

General Definitions. The following definitions define general purpose entities used in describing the actual commands and capabilities of the UDDL. These definitions are presented here because of their frequent use and/or low level nature.

```
alphabetic-character ::=

     A | B | . . . | Z | a | . . . | z | # | - |  |.

name-type ::= alphabetic-character [ name-type ].
site-name ::= name-type.
new-site-name ::= name-type.
new-field-type-id ::= name-type.
new-relation-name ::= name-type.
site-id ::= name-type.
new-function-name ::= name-type.
database-name ::= name-type.
old-database-name ::= name-type.
new-database-name ::= name-type.
field-type-id ::= name-type.
field-name ::= name-type.
relation-name ::= name-type.
function-name ::= name-type.
temp-rel-name ::= name-type.
attribute-name ::= name-type.
attribute-identifier ::= temp-rel-name.attribute-name.

number-character ::=  0 | 1 | 2 | . . . | 9 |.

number ::= number-character [ number ].
real-field-length ::= number.
right-decimal-places ::= number.
field-length ::= number.

decimal-specification ::=

      real-field-length.right-decimal-places.
```

Command Definitions. This definition subsection describes the powers and capabilities of the UDDL. It should be noted that the complete capabilities of the UDDL (create, expand, rename, delete) will not necessarily be user-permitted

operations. The primary purpose of the UDDL is to inform the users and system what comprises the universal view of the data base. If the users are permitted full capability, this could cause security and integrity problems. Imagine a user deleting a data base within the UDB which is comprised of only portions of several other physical data bases. It is for this reason that full UDDL capability is limited to the UDBAC.

Another point requiring discussion is the alternate names definition. This section of a universal relation description is used to inform the user that a given universal relation, for example:

DOCTOR-PATIENT

DOC#	REG#

is physically established in such a manner that to derive certain information, it would be much more efficient for the system if the user were to use a particular alternate name. The heirarchical algorithms in chapter 6 provide a good example. A comment follows each alternate name to convey to the user the type of information that could best be gained through this alternate name.

```
( Create Data Base database-name
  ( [ domain-specification [ domain-specification ] ]
    relation-specification [ relation-specification ]
    user-defined-function [ user-defined-function ] ) )
|
( Expand Data Base database-name
  ( [ domain-specification [ domain-specification ] ]
    [ relation-specification [ relation-specification ] ]
    [ user-defined-function [ user-defined-function ] ] ) )
```

```
    |
  ( Rename Data Base ( rename-database-entry
                       [, rename-database-entry ] ) )
    |
  ( Delete Data Base database-name
    [ ( delete-database-entry [, delete-database-entry ] ] ))

domain-specification ::=

       Domain field-type-id field-type
              [ ( field-length [, NONULL] ) ]

field-type ::=

     Integer | Real | Small Integer | Character |
     Character Var

relation-specification ::=

     Create relation-name
     [ unique-association-specification ]
     [ alternate-name-specification ]
     ( field-defn [, field-defn ]
     Keys are field-name [, field-name ] )

unique-association-specification ::=

      Unique Association between
        attribute-identifier and attribute-identifier

alternate-name-specification ::=

     Alternate names are relation-name comment-field
       [, alternate-name-specification ]

comment-field ::=

     (* alphabetic-character [ alphabetic-character ] *)

field-defn ::=

     [ Unique ]  field-name:  field-type-id |
                 field-type [ ( field-length [, NONULL ] ) ]

user-defined function ::=

     User [ Temporary ] Function  function-name
     [ Input Arguments are ( arg-list: field-defn ) ]
     [ Output Arguments are ( arg-list ) ]
       command-specification (* see UDML *)
     End Function
```

```
arg-list ::=

     attribute-name | relation-name | database-name
     [, arg-list ]

rename-database-entry ::=

     database-name new-database-name |
     field-type-id new-field-type-id |
     relation-name new-relation-name |
     function-name new-function-name

delete-database-entry ::=

     (* nothing - indicates entire data base *) |
     relation-name |
     function-name
```

UDB Medical Data Base (partial) Example:

```
( Create Data Base MEDICAL DATA BASE
(

Domain char-15 Character (15)
Domain char-20 Character (20)
Domain code-type Integer (NONULL)
Domain #type Integer (NONULL)
Domain small-int Small Integer
Domain var-string Character Var


Create HOSPITAL (
     Unique Hospital code:  code-type,
     name:  char-15,
     address:  char-20,
     phone#:  #type,
     # of beds (Small Integer),
     Keys are Hospital code )


Create LAB (
     Unique Lab#:  #type,
     Hospital code:  code-type,
     name:  char-20,
     address:  char-20,
     phone#:  #type,
     Keys are Lab# )
```

```
Create PATIENT-DOCTOR
     Alternate names are ATTENDING DOCTOR
       (*  Treat as normal relation *)
     Alternate names are PAT-ATTD
       (*  List of patients a given doctor is seeing *)
     Alternate names are PATIENTS ATTENDED
       (*  List of patients a given doctor is seeing *)
     Alternate names are ATT-DOC
       (*  List of doctors attending given patient *)
     Alternate names are DOCTORS ATTENDING
       (*  List of doctors attending given patient *)
     Unique Doctor#:  #type,
     Unique Registration#:  #type,
     Keys are All )

)


( Expand Data Base MEDICAL DATA BASE
(  User Function LABS-SERVING-A-HOSPITAL
   Input Arguments are (hosp-name:  code-type)
   Output Arguments are (lab-name:  #type)
   Retrieve
   From  LAB known by L,
         HOSPITAL-LAB known by HL,
         HOSPITAL known by H,
     ( Where hosp-name = H.name and
         H.Hospital code = HL.Hospital code and
         L.Lab# = HL.Lab#
       ( Return (L.name Ordered by Ascending L.name))
  END FUNCTION )
```

Universal Data Manipulation Language

The universal data manipulation language (UDML) specifies the commands that the user and/or UDBAC may execute against the UDB. The approach taken in designing the UDML was to first design a powerful relational DML and then modify it to handle the special cases and problems of the UDB environment. It should be noted that the embedded DDL (as well as deletion and update) capability is limited to the UDBAC. The language is described in two sections: the general definitions section and the command definitions section.

General Definitions. The following definitions are for the general purpose entities used in describing the commands and capabilities of the UDML. They are presented here because of their frequent use and low level nature.

```
alphabetic-character ::=

     A | B | . . . | Z | a | . . . | z | # | - |  |.

name-type ::= alphabetic-character [ name-type ].
site-name ::= name-type.
site-id ::= name-type.
file-name ::= name-type.
field-name ::= name-type.
field-list ::= field-name [, field-list ]
view-name ::= name-type.
old-view-name ::= name-type.
new-view-name ::= name-type.
database-name ::= name-type.
relation-name ::= name-type.
function-name ::= name-type.
temp-rel-name ::= name-type.
attribute-name ::= name-type.
attribute-identifier ::= temp-rel-name.attribute-name.

number-character ::=  0 | 1 | 2 | . . . | 9 |.

number ::= number-character [ number ].
real-field-length ::= number.
right-decimal-places ::= number.
field-length ::= number.

decimal-specification ::=

      real-field-length.right-decimal-places.

constant-value ::= number | name-type.
label ::= number.

system-function ::= MAX | MIN | SUM | COUNT | AVG | UNIQUE .
assign-operator ::= :=.
boolean-operator ::= < | > | <= | >= | = | AND | OR | XOR.
mathematical-operator ::= + | - | / | ** | *.

operator ::=

      assign-operator | boolean-operator |
      mathematical operator
```

Command Definitions. The following definitions describe the powers and capabilities of the UDML.

```
database-command ::=

     ( Access Data Base database-name
     [ view-specification [ view-specification ] ]
     [ command-specification [ command-specification ] ] )

view-specification ::=

     ( Define View [ Temporary ] view-name ( field-list )
       From relation-reference [, relation-reference ]
       As Where selection-criteria ) |

     Delete View ( view-name ) |

     Rename View ( old-view-name new-view-name )

selection-criteria ::=

     ( ( command-specification |
         criteria-specification |
         function-call  )
       [ operator selection-criteria ] )

function-call ::=

    system-function ( ( selection-criteria |
                        attribute-identifier )
                      [ operator target-value ] )
  |
     user-function ( ( input-arg-list ) ( output-arg-list ) )

input-arg-list ::=

     input-attribute [, input-attribute ]

input-attribute ::=

     constant-value | attribute-identifier |
     database-name | relation-name

output-arg-list ::=

     output-attribute [, output-attribute ]

output-attribute ::=

     database-name | attribute-identifier | relation-name
```

```
criteria-specification ::=

     attribute-identifier operator target-value

target-value ::=

     constant-value | function-specification |
     command-specification | attribute-identifier

relation-reference ::=

     relation-name known by temp-rel-name |
     relation-name (temp-rel-name) |
     temp-rel-name known by temp-rel-name [; temp-rel-name ]

command-specification ::=

     ( command-word
       From relation-reference [, relation-reference ]
       Where selection-criteria
       [ action-specification ] )

command-word ::= Retrieve | Update | Delete.

action-specification ::=

     display-command |
     case-command |
     return-command |
     to-command |
     command-specification |
     assignment-operator |
     create-database-operation (* see UDDL *)
     [ action-specification ]

display-command ::=

     ( Display [ File ( file-name ) ]
       ( display-stmt [ display ] ) )

display-stmt ::=

     display-line [ display-line ]

display-line ::=

     [ Unique ] [ label: ] component-expression
                          [, component-expression ]
     [ Ordered by order-criteria ]
```

```
component-expression ::=

     attribute-identifier |
     criteria-specification |
     selection-criteria |
     format-specification

format-specification ::=

     number | LF (* line feed *) | R (* return *) |
     B (* blanks *)
     [, format-specification | format-specification ]

order-criteria ::=

     order-type attribute-identifier |
     [, order-criteria ]

case-command ::=

     ( Case
       case-specification(-1):  case-action(-1)
       case-specification(-2):  case-action(-2)
                                .
                                .
                                .
       case-specification(-n):  case-action(-n)
     [ Otherwise:  case-action(-n+1) ] )

case-specification ::=

     attribute-identifier operator constant-value |
     attribute-identifier operator target-value |
     selection-criteria

case-action ::=  command-specification

return-command ::=

     ( Return ( return-line [ return-line ]
                [ Ordered by order-type order-criteria ] ) )

return-line ::=

     [ Unique ] [ label: ] component-expression
                          [, component-expression ]

order-type ::= Ascending | Descending
```

```
to-command ::=

     ( To attribute-identifier assign-operator new-value |
       To attribute-identifier assign-operator target-value |
       To target-value assign-operator new-value |
       To target-value assign-operator target-value )

direct-reference ::=

     source-value assign-operator target-value

source-value ::=

     attribute-identifier | program-variable

program-variable ::= name-type.

create-database-operaton ::=

     database-specification |
     relation-specificaton |
     domain-specification |
     user-defined-function
```

UMDL Examples

This section of Chapter 7 presents sample queries illustrating some of the various UDML commands. The reader should refer to Appendix G.

1. List all hospitals and their addresses.

```
( Retrieve
  From HOSPITAL known by H,
  ( Return ( H.name, H.address ) ) )
```

2. List all doctors serving in hospitals with over 250 beds.

```
( Retrieve
  From HOSPITAL known by H,
       DOCTOR known by D,
    ( Where H.# of beds > 250 and
      H.Hospital code = D.Hospital code
      ( Return (D.name) ) ) )
```

3. List all doctors serving in hospitals in Tollersville.

```
( Retrieve
  From HOSPITAL known by H,
       DOCTOR known by D,
       HOSPITAL LOCATION known by HL,
       ( Where H.Hospital code = D.Hospital code
         and HL.code = "TLV"
         ( Return (D.name) ) ) )
```

4. List all doctors who attend over 15 patients and list those patients under each doctor's name.

```
( Retrieve
  From DOCTORS known by D,
       DOCTOR-PATIENT known by DP,
  ( Where COUNT (D.Doctor# = DP.Doctor# ) > 15
    Return (D.name, 2RLF,
    ( Retrieve
      From  PATIENT known by P,
      ( Where D.Doctor# = DP.Doctor# and
        P.Registration# = DP.Registration#
        ( Return (P.name ordered by Ascending P.name,
          2RLF) ) ) ) ) )
```

5. Add a 5% pay raise to all staff employees earning over $16,000 and a 10% raise to those earning under $16,000. E shift employees earn an additional 2% pay raise.

```
( Update
  From  STAFF known by S,
   ( Case
       S.salary > 16000:
         ( Case
            S.shift = 'E':  S.salary := S.salary * 1.07,
            Otherwise:  S.salary := S.salary * 1.05 )
       Otherwise:
         ( Case
            S.shift = 'E':  S.salary := S.salary * 1.12,
            Otherwise:  S.salary := S.salary * 1.1 )
    ) )
```

Data Dictionary

A data dictionary is basically a collection of information about information. It is often used to describe entities, activities, processes, and so forth in a system. In the data base context, a data dictionary describes the differ-

ent entities within the data base. It describes the data bases within a system, what relations (records/segments) are contained within each data base, what attributes each relation has, and so forth. In describing each attribute, the data dictionary would indicate what type an attribute is (integer, character, etc.), how many of that type it has (15 characters) if applicable, if the attribute is a key, etc. In the UDB context, the data dictionary takes on even greater responsibility as it must not only describe the contents of the data bases in the UDB but must also describe the data bases. The UDB data dictionary contents described below are concerned with providing sufficient information to perform the UDDL mappings.

Key. The following abbreviations are used in describing the data dictionary.

id - identification number or character string
uatt - universal attribute
urel - universal relation
ufcn - universal function
locn - location (used with ufcn or pfcn to indicate where actual code of ufcn stored at LDBMS)
udom - universal domain
udb - universal data base
ualt - universal alternate (relation) name
oatt - owner attribute
matt - member attribute (matt uniquely assigned to oatt)
pent - physical entity (pseg, prec, prel)
patt - physical attribute
prel - physical relation
pseg - physical segment
prec - physical record
pfcn - physical function
pdb - physical data base
pdom - physcial domain
struct - boolean, does pset in question have structural set selection.
mid - machine id (indicates type of computer DBMS on)

addr - network address of a DBMS
mtype - model type (IMS, DBTG, SYSR)
sid - site id (identifies physical location)
pent - composite for prel, pseg, and prec
seq# - sequence number (IMS)
pfcn - physical function (function defined at LDBMS)
type - attribute type (integer, real, etc)
nonull - is character not allowed to be NULL
pset - DBTG set
own id - DBTG owner record (own id = pseg id)
mem id - DBTG member record (mem id = pseg id)
setsel - set selection (DBTG)
len - length (as in number of characters, etc.)
ret - retention (DBTG)
ins - insertion (DBTG)
par - parent (IMS-same as a pseg)
chd - child (IMS-same as pseg)
rep - replication code (F-fully, P-partially, N-none)
ones- one site (is data base or relation exclusively at one data base or site. Y-Yes, N-No)

UDB UDDL Data Dictionary.

UDB LIST

udb id	udb name	ones

UKEY LIST

uatt id	urelid

UATT LIST

uatt id	uatt name	unique	rep

UFCN LIST

ufcn id	ufcn name	ufcn locn

UDB-UREL LIST

udb id	uatt id

UREL LIST

urel id	urel name	rep	ones

UREL-UATT

urel id	uatt id

UDOM LIST

udom id	udom name	type	len	nonull

UNIQUE ASSOCIATION LIST

urel id	oatt id	matt id

SITE-UDB

udb id	sid

ALTERNATE RELATION LIST

urel id	pset id	ualt name

DOMAIN-UDB

udom id	udb id

SITE LIST

sid	site name	site addr

UFCN INPUT ARGUMENTS

ufcn id	uatt id

UFCN-UDB

udb id	ufcn id

MACHINE LIST

mid	machine name	other info

UFCN OUTPUT ARGUMENTS

ufcn id	uatt id

PREL LIST

prel id	prel name

PSEG LIST

pseg id	pseg name	pseg seq#

PREC LIST

prec id	prec name

PATT LIST

patt id	patt name	unique

PSET LIST

pset id	pset name	own id	mem id	setsel patt

PSET LIST (cont)

ret id	ins id	order id	struct

ORDER LIST

order id	order type

RETENTION LIST

ret id	ret name

INSERTION LIST

ins id	ins name

PFCN LIST

pfcn id	pfcn locn	pdb id

PDOM LIST

pdom id	pdom name	type	len	nonull

SITE MODELS

sid	mtype

SITE MACHINES

sid	mid

PDB MODELS

pdb id	mtype

PDB LIST

pdb id	pdb name

PDB MACHINES

pdb id	mid

PATT-PENT

pent id	patt id

PKEY-PDB

patt id	pent id

PENT-PDB

pent id	pdb id

PDB-PDOM LIST

pdb id	pdom id

PATT-PDOM LIST

patt id	pdom id

PARENT-CHILD

par id	chd id

PFCN INPUT ARGUMENTS

pfcn id	patt id

PFCN OUTPUT ARGUMENTS

pfcn id	patt id

VIII. Results and Conclusions

Introduction

In the preceding seven chapters this thesis has analyzed the problem of trying to permit the effective communication between heterogeneous DBMSs in a distributed environment. During this analysis many issues and problems were raised, discussed, and, if possible, solved or a possible approach indicated. This chapter serves to discuss a few odd issues of the UDB and to note what was accomplished and what needs to be accomplished.

Overview

The first section addresses some of the powers and responsibilities of the UDBAC. The second section discusses the relational model actually used for the UDM. The third section summarizes what was accomplished in this thesis. The fourth section summarizes known deficiencies with the UDB as presented by this thesis. The fifth section lists possible follow-on efforts and the sixth section concludes the chapter and this thesis.

The Powers and Responsibilities of the UDBAC

The role of the UDBAC in the UDB cannot be underestimated. The UDBAC will have the responsibility of formulating policies concerning old data base modification, new data base formation, security, deletions, and updates. The UDBAC will also have to enforce these policies. It will be responsible

for integrating the individual universal representations into one universal view. It is important to note that the UDBAC will have to monitor and control dynamic changes in the underlying data bases. It is difficult to envision at this time that users will be able to make "real time" changes (adding/deleting data bases, structures within a data base, etc.) to their the local data bases. Most likely, the local DBAs for an LDBMS will notify the UDBAC of a new data base being brought up, or a old one deleted, and so forth and the UDBAC will prepare the UDB to accept the new change. It is important to note that not every data base (or portion of a data base) at a local site will necessarily be included in the UDB.

An Augmented Relational Model

As the UDB system was analyzed and developed through this thesis, it became obvious that the relational model, as it stood, would require some modifications to fully satisfy the UDB requirements. The reason for these modifications come from attempting to map nonrelational structures and operations into a relational model. It is thought that it will require even further modifications after the DML requirements are fully analyzed. The present modifications, relatively minor, do not change the basic nature of the relational model, they merely augment it. It is suggested that the additional integrity constraints suggested by Date (3) be fully supported. The following are characteristics of the

the augmented relational model used:

1. Requires only 1NF.
2. Alternate names construct to handle DBTG structural sets.
3. Unique associations clause to alert users to a DBTG fixed retention set.

The following addtional characteristics are anticipated:

1. Support of Domain Integrity (see p. 77, Chapter 4).
2. Support of Immediate Record State Constraint (see p. 77, Chapter 4).
3. Support of Immediate Record Transition Constraint (see p. 77, Chapter 4).
4. Constraint or construct to support DBTG Automatic insertion.
5. Constraint or construct to support DBTG Fixed Retention.

Accomplishments

Although this thesis has not implemented any part of the UDB nor has it really fully investigated all of the issues raised, it does provide a good starting point for further investigation. The following list indicates what was accomplished in this thesis:

1. Literature search of current research into the area of a UDM and/or UDB.
2. An analysis of the requirements for a UDB.
3. The selection of a UDM.
4. An examination of the UDDL mapping issues.
5. Syntax specification for a UDML and UDDL.

Universal Data Model Deficiencies

Although the Relational model was chosen as the best model, of those examined, it is obvious that the UDM, and UDB, as presented in this thesis have several deficiencies or otherwise undesired qualities. It is hoped that these undesired qualities will be eliminated by the time the UDB is actually implemented. The following list summarizes those deficiencies:

1. The UDB is restricted to dealing with one particular implementation of each of the three different models.
2. Each of those three particular implementations (IMS, DBTG, and System R) have "unnatural" constraints imposed upon them by the UDB.
3. The users of the UDB are forced to work in the UDML.
4. Using the UDB may require modifications to the underlying data bases.
5. DML mappings have not been examined to insure that they can be completely supported.
6. The DDL mappings examined have not been fully tested to insure that they are complete and accurate.
7. The relational model used in this thesis could, perhaps, be augmented further to perform better. This reevaluation should be done after the DML requirements have been closely examined.

Follow-on Efforts

As has been pointed out in this thesis, this is an initial examination of the UDB problem. In this thesis, the UDB problem has been analyzed and a direction in which to go decided upon. However, there is a great deal of research yet to be done before the UDB can come even close to a reality.

The following is a list of possible follow-on efforts that need to be done:

1. Study and resolve the DDL DBTG-to-Relational mapping problem with no, or at least a minimal set of, limitations.

2. Study and resolve the DDL IMS-to-Relational mapping problem with no, or at least a minimal set of, limitations.

3. Study and resolve the DML DBTG-to-Relational mapping problem with no, or at least a minimal set of, limitations.

4. Study and resolve the DML IMS-to-Relational mapping problem with no, or at least a minimal set of, limitations.

5. Analyze the UDBA responsibilities and design and implement CAD tools for the UDBA.

6. Design and Implement a complete Universal Data Dictionary for both DML and DDL mapping requirements. Design and Implement a UDML parsing and optimizing algorithm.

7. Study the ER model as a possible replacement for the relational model as the UDM. Essentially, repeat Chapters 5-7 of this thesis for the ER model, but a more in-depth analysis.

Conclusion

Approximately 6 months and 250 pages ago, the goal at the outset was to analyze the UDB problem and determine a direction to take in solving that problem. Both of these goals have been accomplished but a great deal of work remains to be done, if it can. The complexities and scope of this problem present a interesting challenge. The potential gain to be derived from the UDB requires that a continued effort into this area of research be made. Much of the challenge to

be faced by the computer scientists of the 80's and 90's will be how to effectively and efficiently manipulate and process the enormous quantities of information that the modern society now requires to function.

Appendix A:

Acronyms

AFIT - Air Force Institute of Technology

CODASYL - Conference on Data Systems Languages

DBA - Data Base Administrator

DBMS - Data Base Management System

DBTG - Data Base Task Group

DDBMS - Distributed Data Base Management System

DDL - Data Definition Language

DML - Data Manipulation Language

1NF - First Normal Form

GDBMS - Global Data Base Management System

IMS - Information Management System

LDBMS - Local Data Base Management System

LDDL - Local Data Definition Language

LDML - Local Data Manipulation Language

SADT - Structured Analysis and Design Technique

2NF - Second Normal Form

3NF - Third Normal Form

UDB - Universal Data Base

UDBAC - Universal Data Base Administration Center

UDBMS - Universal Data Base Management System

UDDL - Universal Data Definition Language

UDL - Unified Database Language

UDM - Universal Data Model

UDML - Universal Data Manipulation Language

Appendix B:

Glossary of Terms

Action Specification - 1. part of a data base command which specifies what is to done with any data derived from the selection specification.

Alternate Key - 1. A candidate key that is not a primary key.

Atomic - 1. indivisible 2. Cannot be broken down any further. 3. "Nondecomposable so far as the system is concerned (5:86)."

Attribute - 1. A piece of information (e.g. Name, Address, SSN, etc.).

Candidate Key - 1. A set of attributes possessing a unique identification property for a given tuple (5:88).

Casual Data Base User - 1. User who not particularly training the field of computer science and/or data base. 2. a layman.

Child node - 1. Generally used in terms of a heirarchical structure (also known as tree structure) in which elements of the structure are above or below other elements of the structure. A child node is a node or set of nodes immediately below any given node. Good layman example is a family tree where children of a given parent are listed below that parent.

CODASYL - 1. Conference on Data Systems Languages which produced a "standard" for network data bases (see DBTG).

Conceptual data base - 1. "an abstraction of the 'real world' pertinent to an enterprise (12:6)."

Data Base - 1. "Repository for stored data that is both 'integrated' and 'shared' (5:4)." 2. "Collection of stored operational data used by the application system of some particular enterprize (5:7)."

Data Base Administrator - 1. Individual responsible for the creation and maintenance of a data base. 2. "Person (or group of persons) responsible for overall control of database system (5:25).

Data Base Instance - 1. The current contents of a data base.

Data Base Task Group - 1. A working committee under CODASYL which developed the CODASYL "standard" for network data bases called the DBTG network data base.

Data Dictionary - 1. Essentially a data base which contains data about data. 2. description of objects within a data base (5:27).

Data Independence - 1. Immunity of applications to change in storage structure and access strategy (5:13). 2. A change in how the data is physically stored or accessed will not require (significant) changes in any applications.

Data Integrity - 1. Data Integrity is where the data within a data base is accurate. An example of a lack of integrity would be where there is an "inconsistency between two entries representing the same 'fact' (5:11)."

Data Base Management System - 1. A group of software that allows one or more persons to use and/or modify the contents of a data base (12:1).

Data Model - 1. A method of describing a database (12:18). 2. Consists of two elements. "A mathematical notation for expressing data and relationships, and operations on the data that serve to express queries and other manipulation of the data (12:18)."

Distributed Data Base Management System - 1. Data base that is not stored entire-at one physical location but is actually stored at several different locations connected by a computer network.

Data Definition Language - 1. language used to describe the objects within a data base. 2. "High-level language enabling one

to describe the conceptual data-base in terms of a data model (12:6)."

Data Manipulation Language - 1. language used to manipulate objects within a data base.

Domain - 1. a pool of values from which actual values of a column (in a relational table) are drawn (5:65).

Foreign Key - 1. An attribute in a tuple for which it is not a primary key for that relation or record, but is for some other record or relation.

Global Data Base Management System - 1. Another name for a distributed data base management system.

Heterogeneous - 1. Different. 2. In data base context, indicates that the data base systems in question use different data models.

Homogeneous - 1. Same. 2. In data base context, indicates that the data base systems in question use the same data model.

IMS - 1. Information Management System developed by IBM using a heirarchical model.

K - 1. Alphabetic representation for the number 2 raised to tenth power. 2. 1024

Local Data Base Management System - 1. Another name for a normal DBMS. Used in the UDB context to distinguish from a given local system and any DBMS on the network.

Node - 1. In the data base context, a level or group of records in a heirarchical or network data base.

Nonprocedural - 1. In the data base context, describes a language in which the user does not specify how the data base is to access the desired information. Instead the user specifies what selection criteria are to be used in determining what data is to be returned.

Parent Node - 1. Generally used in terms of a heirarchical structure (also known as tree structure) in in which elements of the structure are above or below other elements of the structure. A par-

ent node is a node which is immediately above any other given node. A good layman example is a family tree where the parent(s) of any given child(ren) would be listed above that child.

Primary Key - 1. An attribute whose distinct value will uniquely identify a given tuple from all other tuples in the relation or record.

Procedural - 1. In the data base context, describes a language in which the user must specify how the data base is to access the information the user desires.

Query - 1. A question directed to a data base system concerning the contents of that data base.

Response Time - 1. Time it takes to get a response or to accomplish a computer function.

Root Node - 1. Generally used in terms of a heirarchical structure (also known as tree structure) in which elements of the structure are above or below other elements of the structure. A root node is the topmost node in the structure. It has no parent node (see parent node).

Schema - 1. Another name for data base intension.

Selection Specification - 1. Part of a data base command which specifies what information is desired.

Structured Analysis and Design Technique - 1. A software engineering technique developed by a company called Softech. SADT is used to develop computer systems.

Tuple - 1. A term used in relational data bases to describe a row in a relational table. 2. In the context of other definitions contained within this glossary, it will also refer a record which is a term used when discussing network and/or heirarchical data bases. This is used to avoid confusion when discussing a entry within a data base (tuple or record) and, for instance, a DBTG Record type.

Universal Data Base - 1. Data base designed to allow the communication, in a DDBMS network, of heterogeneous DBMS.

Universal Data Base
Administration Center - 1. An organization whose function is to perform the traditional DBA functions for the UDB.

Universal Data Base
Management System - 1. The DBMS to be developed for the UDB.

Universal Data Definition
Language - 1. DDL to be developed for the UDB.

Universal Data Manipulation
Language - 1. DML to be developed for the UDB.

User - 1. A person, organization, or even another computer which are using a given computer system.

View - 1. "an abstract model of a portion of the conceptual data base (12:7)."

Appendix C:

Medical Data Base Information

(Partial)

Source: The idea and a portion of the information presented below are derived from Tsichritzis and Lochovsky's book, Data Models (9).

New Salisbury: System R (relational)
Tollersville: CODASYL DBTG (network)
Bollington: IMS (heirarchical)

Hospital Information

HC	Name	Address	Phone#	#Beds
New Salisbury:				
22	Doctors	45 Brunswick	923-5411	412
13	Central	333 Sherbourne	964-4264	502
45	Childrens	555 University	597-1500	845
18	General	101 College	595-3111	987
Tollersville:				
10	Southside	15 Old Main	666-4556	234
54	Northside	342 N. Broad	333-9876	987
77	Memorial	22 E. Kaufman	333-1212	450
Bollington:				
03	Kempton's	145 S. Main	345-2323	145
98	Mercy	12 N. Fairfield	543-3232	650
96	St. Paul's	187 S. Mattix	347-9999	234

Lab Information

Lab #	Name	Address	Phone#
New Salisbury:			
56	Alpha	18 Kipling	929-9611
84	Nucro	62 Lyons	368-9703
16	Atcon	14 Main	532-4453
42	Clini	55 King	447-6448
Tollersville:			
11	Huber	123 W. Fudenburg	567-3344
24	Sera	1829 W. Main	234-5678
76	Whitmer	3425 S. Dixie	239-1111
47	Nucro	62 Lyons, N. Sal.	368-9703

Bollington:

90	Blue Rex	12 Altern	321-4418
34	Samaritan	478 Salem Dr.	789-4572

Hospital-Lab Information

New Salisbury:		Tollersville:		Bollington:	
HC	Lab#	HC	Lab#	HC	Lab#
22	56	10	11	03	90
22	84	54	76	98	34
13	16	77	11	98	90
18	56	77	24	96	90
45	16	77	76		
18	84	10	24		
18	16	54	24		
13	42	10	47		
18	42				

Ward Information

HC	Ward Code	Name	# of beds
New Salisbury:			
22	1	Recovery	10
13	3	Intensive Care	21
22	6	Psychiatric	118
45	4	Cardiac	55
22	2	Maternity	34
13	6	Psychiatric	67
18	3	Intensive Care	10
45	1	Recovery	13
18	4	Cardiac	53
45	2	Maternity	24
Tollersville:			
10	1	Recovery	50
10	3	Intensive Care	10
10	4	Cardiac	25
10	5	Cancer	60
54	2	Maternity	20
54	6	Psychiatric	15
77	5	Cancer	45
Bollington:			
03	2	Maternity	20
98	4	Cardiac	25
98	5	Cancer	25
96	1	Recovery	10
96	2	Maternity	25
96	6	Psychiatric	15
96	4	Cardiac	35
03	1	Recovery	20

Staff Information

HC	Ward Code	Employee#	Name	Duty	Shift	Salary
New Salisbury:						
22	6	1009	Homes D.	Nurse	M	18500
13	6	3754	Delagi B.	Nurse	A	17400
22	6	8422	Bell G.	Orderly	M	12600
22	2	9901	Newport C.	Intern	M	17000
45	4	1280	Anderson R.	Intern	E	17000
22	1	6065	Ritchie G.	Nurse	E	20200
13	6	3106	Hughes J.	Orderly	A	13500
45	1	8526	Frank H.	Nurse	A	19400
18	4	6357	Karplus W.	Intern	M	18300
22	1	7379	Colony R.	Nurse	M	16300
Tollersville:						
10	1	3264	McDay, R.	Intern	E	16500
54	6	8472	Stone, B.	Nurse	A	18500
77	5	8300	Turner, C.	Intern	M	17400
10	3	2321	Simpson, D.	Nurse	A	15000
54	2	1111	Griffth, D.	Orderly	M	19300
54	2	2321	Laud, C.	Nurse	A	20000
77	5	8598	Hall, D.	Orderly	M	17600
10	4	4587	Summer, S.	Intern	A	18000
77	5	3322	Jankus, L.	Nurse	A	21000
54	2	0034	Donnely, P.	Orderly	M	17100
Bollington:						
03	2	8733	Bechey, M.	Nurse	M	15500
98	5	2318	Beail, S.	Orderly	A	16700
98	4	6667	Hagan, K.	Intern	E	16000
96	1	4348	Dixon, F.	Intern	A	15800
03	1	0934	Mesker, D.	Orderly	E	16700
98	5	0923	Hyre, K.	Nurse	M	17700
96	6	4567	Foy, C.	Nurse	A	18000
96	4	1129	Stead, R.	Orderly	E	17500

Doctor Information

HC	Doctor#	Name	Specialty
New Salisbury:			
45	607	Ashby W.	Pediatrics
18	585	Miller G.	Gynecology
22	453	Glass D.	Pediatrics
13	435	Lee A.	Cardiology
45	522	Adams C.	Neurology
22	398	Best K.	Urology
18	982	Russ J.	Cardiology
22	386	Stone C.	Psychiatry

Tollersville:

77	324	Pierce, J.	Gynecology
10	542	Kermit, S.	Urology
77	623	Bumble, F.	Neurology
54	111	Morge, C.	Surgery
54	607	Ashby, W.	Pediatrics
10	110	Kildare, D.	Surgery
54	652	Welby, M.	Urology
77	342	Jackson, M.	Gynecology

Bollington:

03	233	Adams, C.	Neurology
98	454	Coyle, E.	Psychiatry
96	213	Rup, D.	Urology
96	310	Rore, T.	Surgery
03	222	DePuso, J.	Pediatrics
98	419	Sutten, P.	Gynecology
03	100	Yast, E.	Psychiatry
98	035	Tunwe, B.	Surgery

Patient-Doctor Information

New Salisbury		Tollersville		Bollington	
Doctor#	REG#	Doctor#	REG#	Doctor#	REG#
607	74537	623	29388	035	88719
435	18004	111	56473	310	10459
522	56473	342	36455	419	83425
386	36658	607	36455	100	64822
453	18004	542	29388	454	10959
982	24024	110	67743	419	83425
585	59076	324	46384	233	34221
398	74835	111	56473	035	69582
386	10995	623	38702	222	34221
607	64823				

Patient Information

REG#	Name	Address	Birthdate	Sex	SSN
New Salisbury:					
63827	Rasky P.	60 Bathhurst	Jun 1 1945	M	100973253
36658	Domb B.	55 Patina	Apr 8 1954	M	660657471
74537	Gettel, B.	73 Dixie, B1	May 15 1967	F	473636363
64823	Fraser A.	11 Massey	May 3 1960	F	985201776
74835	Bower E.	15 Ontario	Oct 16 1933	M	654811767
56473	Walker, S.	21 Tatonie T1	Dec 1 1945	M	498562224
18004	Shiu W.	14 Ivy	Jan 22 1916	F	914991452
59076	Miller G.	80 Lawton	Jun 4 1971	F	611969044
24024	Fourie M.	40 Donora	Jul 9 1966	F	321790059
10995	Lista M.	58 Olsen	Nov 7 1963	M	980862482
39217	Birze H.	51 Dallas	Aug 20 1958	M	740294390

Tollersville:

24568	Rice D.	22 Hall Ave	Oct	23	1935	F	485756383
46384	Brumbaugh M.	42 Brown	Dec	14	1940	F	371649500
38702	Neal R.	65 Halsey	Nov	3	1949	F	380010217
74835	Bower, E.	15 Ontario NS	Oct	16	1933	M	654811767
29388	Blakely, J.	123 W. Katz	Nov	12	1961	M	034948575
36455	Warden, D.	23 N. Dinky St.	Jun	17	1963	F	756646463
35467	Donn, P.	89 W. Third	Oct	9	1978	F	045958588
67743	Stoen, W.	345 Willkie	Aug	26	1964	M	746535222
56473	Walker, S.	21 Tatonie	Dec	1	1945	M	498562224

Bollington:

10959	Kirk, J.	46 E. Prize	Jun	11	1940	M	847666632
34221	Heller, D.	453 W. Virg	Nov	23	1965	F	317837444
56473	Walker, S.	21 Tatonie TL	Dec	1	1945	M	498562224
83425	Marvin, M.	43 Indy St.	Jul	4	1960	F	946353755
74537	Gettel, B.	73 Dixie	May	15	1967	F	473636363
69582	Honna, P.	429 Penn St.	Apr	20	1942	F	973047057
37719	Riker, C.	519 Moyer	Nov	01	1892	M	168725816
68746	Dink, L.	216 Posch	Feb	04	1937	M	581398427
88719	DeLong, P.	600 Scenic	May	29	1961	F	569712534
64822	Cruz, M.	214 Organ	Dec	25	1932	F	249463746

Appendix D:

Relational Version of Medical Data Base

[9:96-99]

HOSPITAL

Hospital code	*Name*	*Address*	*Phone#*	*# of beds*
22	Doctors	45 Brunswick	923-5411	412
13	Central	333 Sherbourne	964-4264	502
45	Childrens	555 University	597-1500	845
18	General	101 College	595-3111	987

WARD

Hospital code	*Ward code*	*Name*	*# of beds*
22	1	Recovery	10
13	3	Intensive Care	21
22	6	Psychiatric	118
45	4	Cardiac	55
22	2	Maternity	34
13	6	Psychiatric	67
18	3	Intensive Care	10
45	1	Recovery	13
18	4	Cardiac	53
45	2	Maternity	24

STAFF

Hospital code	*Ward code*	*Employee#*	*Name*	*Duty*	*Shift*	*Salary*
22	6	1009	Holmes D.	Nurse	M	18500
13	6	3754	Delagi B.	Nurse	A	17400
22	6	8422	Bell G.	Orderly	M	12600
22	2	9901	Newport C.	Intern	M	17000
45	4	1280	Anderson R.	Intern	E	17000
22	1	6065	Ritchie G.	Nurse	E	20200
13	6	3106	Hughes J.	Orderly	A	13500
45	1	8526	Frank H.	Nurse	A	19400
18	4	6357	Karplus W.	Intern	M	18300
22	1	7379	Colony R.	Nurse	M	16300

DOCTOR

Hospital code	*Doctor#*	*Name*	*Specialty*
45	607	Ashby W.	Pediatrics
18	585	Miller G.	Gynecology
22	453	Glass D.	Pediatrics
13	435	Lee A.	Cardiology
45	522	Adams C.	Neurology
22	398	Best K.	Urology
18	982	Russ J.	Cardiology
22	386	Stone C.	Psychiatry

PATIENT

Registration#	Name	Address	Birthdate	Sex	SSN
63827	Rasky P.	60 Bathurst	Jun 1 1945	M	100973253
36658	Domb B.	55 Patina	Apr 8 1954	M	660657471
64823	Fraser A.	11 Massey	May 3 1960	F	985201776
74835	Bower E.	15 Ontario	Oct 16 1933	M	654811767
18004	Shiu W.	14 Ivy	Jan 22 1916	F	914991452
59076	Miller G.	80 Lawton	Jun 4 1971	F	611969044
24024	Fourie M.	40 Donora	Jul 9 1966	F	321790059
10995	Lista M.	58 Olsen	Nov 7 1963	M	980862482
39217	Birze H.	51 Dallas	Aug 20 1958	M	740294390
38702	Neal R.	65 Halsey	Nov 3 1949	F	380010217

LAB

Lab#	Name	Address	Phone#
56	Alpha	18 Kipling	929-9611
84	Nucro	62 Lyons	368-9703
16	Atcon	14 Main	532-4453
42	Clini	55 King	447-6448

HOSPITAL LAB

Hospital code	Lab#
22	56
22	84
13	16
18	56
45	16
18	84
18	16
13	42
18	42

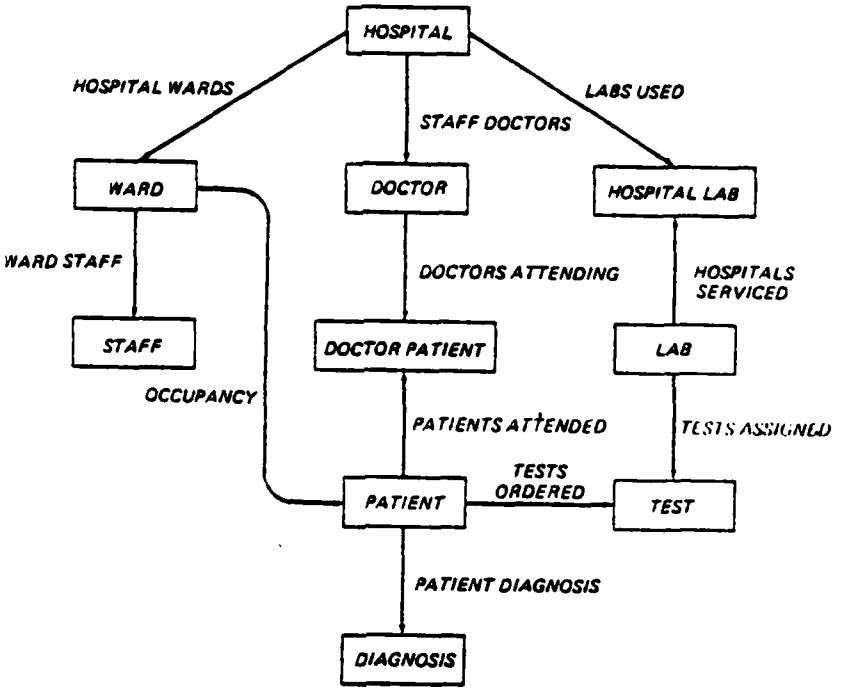
```
CREATE TABLE HOSPTIAL:
  Hospital code (INTEGER,NONULL),
  Name (CHAR (15)),
  Address (CHAR (20)),
  Phone# (CHAR (7)),
  # of beds (SMALLINT)

CREATE TABLE HOSPITAL LAB:
  Hospital code (INTEGER,NONULL),
  Lab# (INTEGER,NONULL)

CREATE TABLE ATTENDING DOCTOR:
  Doctor# (INTEGER,NONULL),
  Registration# (INTEGER,NONULL)
```

```
CREATE TABLE OCCUPANCY:
  Hospital code (INTEGER,NONULL),
  Ward code (INTEGER,NONULL),
  Registration# (INTEGER,NONULL),
  Bed# (INTEGER,NONULL)

CREATE TABLE WARD:
  Hospital code (INTEGER,NONULL),
  Ward code (INTEGER,NONULL),
  Name (CHAR (15)),
  # of beds (SMALLINT)

CREATE TABLE STAFF:
  Hospital code (INTEGER,NONULL),
  Ward code (INTEGER,NONULL),
  Employee# (INTEGER,NONULL),
  Name (CHAR (20)),
  Duty (CHAR (*)),
  Shift (CHAR (10)),
  Salary (DECIMAL (7,2)

CREATE TABLE DOCTOR:
  Hospital code (INTEGER,NONULL),
  Doctor# (INTEGER,NONULL),
  Name (CHAR (20)),
  Specialty (CHAR (*))

CREATE TABLE PATIENT:
  Registration# (INTEGER,NONULL),
  Name (CHAR (20)),
  Address (CHAR (20)),
  Birthdate (CHAR (8)),
  Sex (CHAR (1)),
  SSN (INTEGER,NONULL)

CREATE TABLE DIAGNOSIS:
  Registration# (INTEGER,NONULL),
  Diagnosis code (INTEGER,NONULL),
  Diagnosis type (CHAR (*)),
  Complications (CHAR (*)),
  Precautionary info (CHAR (*))

CREATE TABLE LAB:
  Lab# (INTEGER,NONULL),
  Name (CHAR (20)),
  Address (CHAR (20)),
  Phone# (CHAR (7))
```

```
CREATE TABLE TEST:
  Registration# (INTEGER,NONULL),
  Lab# (INTEGER,NONULL),
  Test code (INTEGER,NONULL),
  Type (CHAR (20)),
  Date ordered (CHAR (8)),
  Time ordered (CHAR (8)),
  Specimen/order# (INTEGER),
  Status (CHAR (*))
```

Appendix E: DBTG Version of Medical Data Base

[9:121-125]



HOSPITAL(Hospital code, Name, Address, Phone#, # of beds)
WARD(Ward code, Name, # of beds)
STAFF(Employee#, Name, Duty, Shift, Salary)
DOCTOR(Doctor#, Name, Specialty)
DOCTOR PATIENT(Doctor#, Registration#)
PATIENT(Registration#, Bed#, Name, Address, Birthdate,
Sex, SSN)
DIAGNOSIS(Diagnosis code, Diagnosis type, Complications,
Precautionary Info)
HOSPITAL LAB(Hospital code, Lab#)
LAB(Lab#, Name, Address, Phone#)
TEST(Specimen/order#, Type, Data ordered, Time ordered,
Testcode, Status)

```
RECORD NAME IS HOSPITAL
  DUPLICATES ARE NOT ALLOWED FOR Hospital code.
     Hospital code TYPE IS FIXED 6
     Name TYPE IS CHARACTER 15
     Address TYPE IS CHARACTER 20
     Phone# TYPE IS CHARACTER 7
     # of beds TYPE IS FIXED 4
```

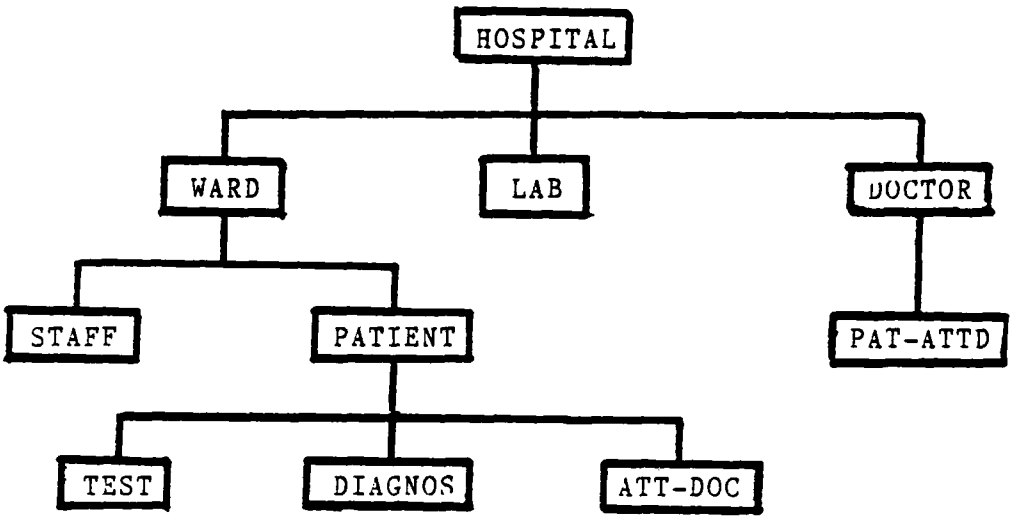
```
RECORD NAME IS WARD
  DUPLICATES ARE NOT ALLOWED FOR Ward code.
     Ward code TYPE IS FIXED 6
     Name TYPE IS CHARACTER 15
     # of beds TYPE IS FIXED 4

RECORD NAME IS STAFF
  DUPLICATES ARE NOT ALLOWED FOR Employee#.
     Employee# TYPE IS FIXED 6
     Name TYPE IS CHARACTER 20
     Duty TYPE IS CHARACTER 15
     Shift TYPE IS CHARACTER 10
     Salary TYPE IS FIXED 5 2

RECORD NAME IS DOCTOR
  DUPLICATES ARE NOT ALLOWED FOR Doctor#
     Doctor# TYPE IS FIXED 6
     Name TYPE IS CHARACTER 20
     Specialty TYPE IS CHARACTER 20

RECORD NAME IS DOCTOR PATIENT
  DUPLICATES ARE NOT ALLOWED FOR Doctor# Registration#.
     Doctor# TYPE IS FIXED 6
     Registration# TYPE IS FIXED 6

RECORD NAME IS PATIENT
  DUPLICATES ARE NOT ALLOWED FOR Registration#
  DUPLICATES ARE NOT ALLOWED FOR SSN.
     Registration# TYPE IS FIXED 6
     Bed# TYPE IS FIXED 4
     Name TYPE IS CHARACTER 20
     Address TYPE IS CHARACTER 20
     Birthdate TYPE IS CHARACTER 8
     Sex TYPE IS CHARACTER 1
         CHECK IS VALUE 'F', 'M'
     SSN TYPE IS FIXED 6

RECORD NAME IS DIAGNOSIS
     Diagnosis code TYPE IS FIXED 6
     Diagnosis type TYPE IS CHARACTER 25
     Complications TYPE IS CHARACTER 25
     Precautionary Info TYPE IS CHARACTER 40

RECORD NAME IS HOSPITAL LAB
  DUPLICATES ARE NOT ALLOWED FOR Hospital code. Lab#.
     Hospital code TYPE IS FIXED 6
     Lab# TYPE IS FIXED 6
```

```
RECORD NAME IS LAB
  DUPLICATES ARE NOT ALLOWED FOR Lab#.
     Lab# TYPE IS FIXED 6
     Name TYPE IS CHARACTER 20
     Address TYPE IS CHARACTER 20
     Phone# TYPE IS CHARACTER 7

RECORD NAME IS TEST
     Test code TYPE IS FIXED 6
     Type TYPE IS CHARACTER 20
     Date ordered TYPE IS CHARACTER 8
     Time ordered TYPE IS CHARACTER 4
         CHECK IS VALUE 0 THRU 2400.
     Specimen/order# TYPE IS FIXED 6
     Status TYPE IS FIXED 15

SET NAME IS LABS USED.
  OWNER IS HOSPITAL
    ORDER IS NEXT.
  MEMBER IS HOSPITAL LAB
    INSERTION IS AUTOMATIC RETENTION IS FIXED
    SET SELECTION IS BY
      STRUCTURAL Hospital code = Hospital code.

SET NAME IS OCCUPANCY.
  OWNER IS WARD
    ORDER IS SYSTEM DEFAULT.
  MEMBER IS PATIENT
    INSERTION IS MANUAL RETENTION IS OPTIONAL
    SET SELECTION IS BY VALUE OF Ward code.

SET NAME IS STAFF DOCTORS.
  OWNER IS HOSPITAL
    ORDER IS SORTED BY DEFINED KEYS
    DUPLICATES ARE NOT ALLOWED.
  MEMBER IS DOCTOR
    INSERTION IS MANUAL RETENTION IS OPTIONAL
    SET SELECTION IS BY VALUE OF Hospital code.

SET NAME IS DOCTORS ATTENDING
  OWNER IS DOCTOR
    ORDER IS NEXT.
  MEMBER IS DOCTOR PATIENT
    INSERTION IS AUTOMATIC RETENTION IS FIXED
    SET SELECTION VALUE IS BY STRUCTURAL Doctor# = Doctor#

SET NAME IS PATIENTS ATTENDED.
  OWNER IS PATIENT
    ORDER IS NEXT.
  MEMBER IS DOCTOR PATIENT
    INSERTION IS AUTOMATIC RETENTION IS FIXED
    SET SELECTION IS BY STRUCTURAL
```

```
        Registration# = Registration

SET NAME IS PATIENT DIAGNOSIS
  OWNER IS PATIENT
    ORDER IS LAST.
  MEMBER IS DIAGNOSIS
    INSERTION IS AUTOMATIC RETENTION IS FIXED
    SET SELECTION IS BY VALUE OF Registration#.

SET NAME IS TESTS ORDERED.
  OWNER IS PATIENT
    ORDER IS FIRST.
  MEMBER IS TEST
    INSERTION IS AUTOMATIC RETENTION IS FIXED.

SET NAME IS TEST ASSIGNED.
  OWNER IS LAB
    ORDER IS LAST.
  MEMBER IS TEST
    INSERTION IS AUTOMATIC RETENTION IS FIXED.

SET NAME IS HOSPITALS SERVICED
  OWNER IS LAB
    ORDER IS NEXT.
  MEMBER IS HOSPITAL LAB
    INSERTION IS AUTOMATIC RETENTION IS FIXED
    SET SELECTION IS BY STRUCTURAL Lab# = Lab#.

SET NAME IS HOSPITAL WARDS
  OWNER IS HOSPITAL
    ORDER IS SORTED BY DEFINED KEYS
    DUPLICATES ARE NOT ALLOWED.
  MEMBER IS WARD
    INSERTION IS AUTOMATIC RETENTION IS FIXED
    SET SELECTION IS BY VALUE OF Hospital code.

SET NAME IS WARD STAFF
  OWNER IS WARD
    ORDER IS SORTED BY DEFINED KEYS
    DUPLICATES ARE NOT ALLOWED.
  MEMBER IS STAFF
    INSERTION IS AUTOMATIC RETENTION IS MANDATORY
    SET SELECTION IS BY VALUE OF Ward code.
```

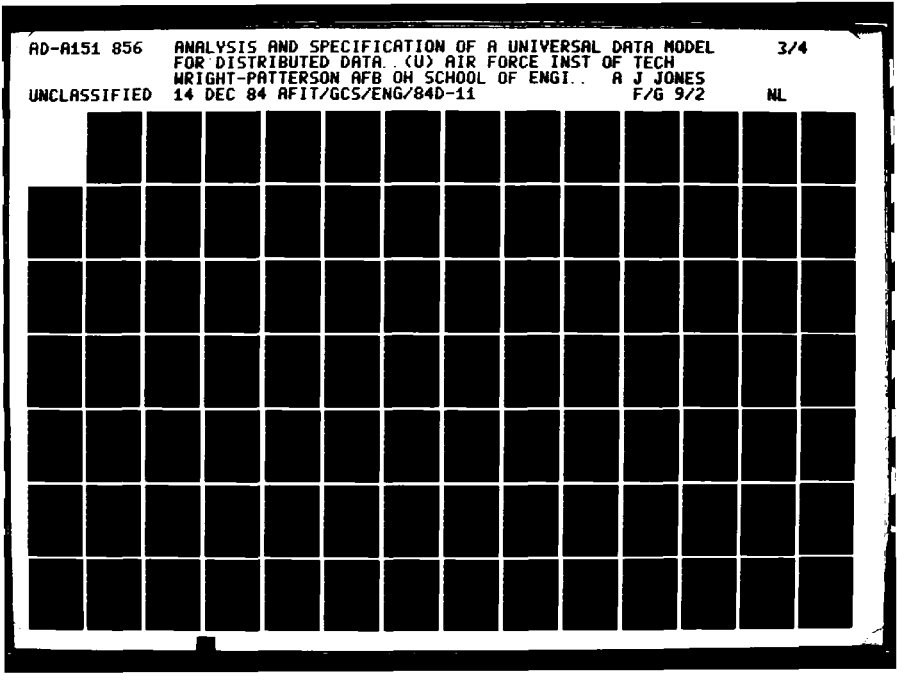
Appendix F:

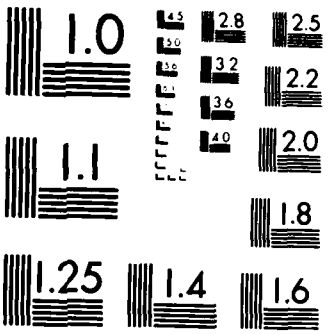
IMS Version of Medical Data Base



HOSPITAL(Hospcode, name, address, phone#, #ofbeds)
LAB(Lab#, name, address, phone#)
WARD(Wardcode, name, #ofbeds)
STAFF(Empl#, name, duty, shift, salary)
PATIENT(Reg#, bed#, name, address, birthdate, sex, SSN)
DOCTOR(Doctor#, name, speclty)
PAT-ATTD(Reg#)
ATT-DOC(Doctor#)
TEST(spc/ord#, lab#, type, dateordr, timeordr, testcode,
status)
DIAGNOS(Diagcode, diagtype, complns, precinfo)

```
 1  PCB      TYPE=DB,DBDNAME=MEDDBD,KEYLEN=15
 2  SENSEG   NAME=HOSPITAL,PROCOPT=G
 3  SENSEG   NAME=WARD,PARENT=HOSPITAL,PROCOPT=G
 4  SENSEG   NAME=LAB,PARENT=HOSPITAL,PROCOPT=G
 5  SENSEG   NAME=DOCTOR,PARENT=HOSPITAL,PROCOPT=GIRD
 6  SENSEG   NAME=STAFF,PARENT=WARD,PROCOPT=GIRD
 7  SENSEG   NAME=PATIENT,PARENT=WARD,PROCOPT=GIRD
 8  SENSEG   NAME=PAT-ATTD,PARENT=DOCTOR,PROCOPT=GIRD
 9  SENSEG   NAME=TEST,PARENT=PATIENT,PROCOPT=GIRD
10  SENSEG   NAME=DIAGNOS,PARENT=PATIENT,PROCOPT=GIRD
11  SENSEG   NAME=ATT-DOC,PARENT=PATIENT,PROCOPT=GIRD
```


AD-A151 856 ANALYSIS AND SPECIFICATION OF A UNIVERSAL DATA MODEL 3/4
FOR DISTRIBUTED DATA..(U) AIR FORCE INST OF TECH
WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. A J JONES
UNCLASSIFIED 14 DEC 84 AFIT/GCS/ENG/84D-11 F/G 9/2 NL



```
 1   DBD       NAME=MEDDBD
 2   SEGM      NAME=HOSPITAL,BYTES=48
 3   FIELD     NAME=Hospcode, BYTES=3,START=1
 4   FIELD     NAME=name,BYTES=15,START=4
 5   FIELD     NAME=address,BYTES=20,START=19
 6   FIELD     NAME=phone#,BYTES=7,START=40
 7   FIELD     NAME=#ofbeds,BYTES=2,START=47
 8   SEGM      NAME=WARD,BYTES=20
 9   FIELD     NAME=Wardcode,BYTES=3,START=1
10   FIELD     NAME=name,BYTES=15,START=4
11   FIELD     NAME=#ofbeds,BYTES=2,START=19
12   SEGM      NAME=LAB,BYTES=50
13   FIELD     NAME=Lab#,BYTES=3,START=1
14   FIELD     NAME=name,BYTES=20,START=4
15   FIELD     NAME=address,BYTES=20,START=24
16   FIELD     NAME=phone#,BYTES=7,START=44
17   SEGM      NAME=DOCTOR,BYTES=83
18   FIELD     NAME=Doctor#,BYTES=3,START=1
19   FIELD     NAME=name,BYTES=20,START=4
20   FIELD     NAME=speclty,BYTES=60,START=24
21   SEGM      NAME=STAFF,BYTES=59
22   FIELD     NAME=Empl#,BYTES=3,START=1
23   FIELD     NAME=name,BYTES=20,START=4
24   FIELD     NAME=duty,BYTES=30,START=24
25   FIELD     NAME=shift,BYTES=1,START=54
26   FIELD     NAME=salary,BYTES=5,START=55
27   SEGM      NAME=PATIENT,BYTES=64
28   FIELD     NAME=Reg#,BYTES=3,START=1
29   FIELD     NAME=bed#,BYTES=3,START=4
30   FIELD     NAME=name,BYTES=20,START=7
31   FIELD     NAME=address,BYTES=20,START=27
32   FIELD     NAME=birthdate,BYTES=8,START=47
33   FIELD     NAME=sex,BYTES=1,START=48
34   FIELD     NAME=SSN,BYTES=9,START=49
35   SEGM      NAME=PAT-ATTD,BYTES=5
36   FIELD     NAME=Reg#,BYTES=5,START=1
37   SEGM      NAME=TEST,BYTES=107
38   FIELD     NAME=Testcode,BYTES=6,START=1
39   FIELD     NAME=Lab#,BYTES=3,START=7
40   FIELD     NAME=type,BYTES=20,START=10
41   FIELD     NAME=dateordr,BYTES=8,START=30
42   FIELD     NAME=timeordr,BYTES=4,START=38
43   FIELD     NAME=spc/ord#,BYTES=6,START=42
44   FIELD     NAME=status,BYTES=60,START=48
45   SEGM      NAME=DIAGNOS,BYTES=186
46   FIELD     NAME=Diagcode,BYTES=6,START=1
47   FIELD     NAME=diagtype,BYTES=60,START=7
48   FIELD     NAME=complns,BYTES=60,START=67
49   FIELD     NAME=precinfo,BYTES=60,START=127
50   SEGM      NAME=ATT-DOC,BYTES=3
51   FIELD     NAME=Doctor#,BYTES=3,START=1
```

Appendix G:

UDB Version of Sample Data Base

```
( Create Data Base MEDICAL DATA BASE
(

Domain char-15 character(15)
Domain char-20 character(20)
Domain code-type integer(NONULL)
Domain #type integer(NONULL)
Domain small-int Small Integer
Domain var-string Character Var


Create HOSPITAL (
     Unique Hospital code:  code-type,
     Name:  char-15,
     Address:  char-20,
     Phone#: #type,
     # of beds (Small Integer),
     Keys are Hospital code )

Create DOCTOR PATIENT (
     Unique Association between Doctor#
          and Registration#
     Alternate names are DOCTORS ATTENDING
     (* doctors attending a given patient *)
     Alternate names are PATIENTS ATTENDED
     (* patients a doctor is attending *)
     Doctor#:  #type,
     Registration#:  #type,
     Keys are Doctor#, Registration#)

Create HOSPITAL LOCATION (
     Unique Hospital code:  code-type,
     Location code:  code-type,
     Keys are Hospital code)

Create LAB LOCATION (
     Unique Lab#:  #type,
     Location code:  code-type,
     Keys are Lab#, Location code)

Create PATIENT LOCATION (
     Unique Registration#:  #type,
     Location code:  code-type,
     Keys are Registration#, Location code)
```

```
Create Ward (
     Unique Association between Hospital code
          and Ward Code
     Hospital Code:  code-type,
     Unique Ward Code:  code-type,
     Name:  char-15,
     # of beds:  small-int,
     Keys are Hospital Code, Ward Code)

Create LAB (
     Unique Lab#:  #type,
     Hospital code: code-type,
     Name:  char-20,
     Address:  char-20,
     Phone#:  #type,
     Keys are Lab# )

Create Staff (
     Hospital code:  code-type,
     Ward Code:  code-type,
     Unique Employee#:  #type,
     Name:  char-20,
     Duty:  var-string,
     Shift:  Character (10),
     Salary:  Real (7.2),
     Keys are Hospital Code, Ward Code, Employee#)

Create DOCTOR (
     Hospital code:  code-type,
     Unique Doctor#:  #type,
     Name:  char-20,
     Specialty:  var-string,
     Keys are Doctor#, Hospital code)

Create PATIENT (
     Unique Registration#:  #type,
     Ward code:  code-type,
     Name:  char-20,
     Address:  char-20,
     Bed#:  #type,
     Birthdate:  Character (8),
     Sex:  Character (1),
     SSN:  #type,
     Keys are Registration#, Ward code)

Create DIAGNOSIS (
     Registration#:  #type,
     Diagnosis code:  code-type,
     Diagnosis type:  var-string,
     Complications:  var-string,
     Precautionary info:  var-string,
     Keys are Diagnosis code, Registration#)
```

```
Create TEST (
     Registration#:  #type,
     Lab#:  #type,
     Test code:  code-type,
     Type:  char-20,
     Date ordered:  Character (8),
     Time ordered:  Character (4),
     Specimen/order#:  Integer,
     Status:  var-string,
     Keys are Test code, Registration#)
```

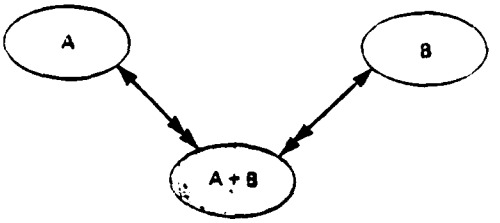
Appendix H:

The Canonical Synthesis Process

[10:249-251]

"1. Take the first user's view of data and draw it in the form of a bubble chart--a graph with point-to-point directed links between single data items, representing associations of the two types: 1 and M.

Where a concatenated key is used, draw this as one bubble, and draw the compound data items of the concatenated key as separate bubbles, thus:



Check that the representation avoids hidden transitive dependencies. Where a concatenated key data item has been used, ensure that all single-arrow links from it go to data items which are dependent on the full concatenated key, not merely part of it. In other words, ensure that the representation of the user's view is in third normal form.

Otherwise, draw only the association that concern this user.

2. Take the next user's view, representing it as above. Merge it into the graph. Check for any synonyms or homonym, removing them if they appear.

3. In the resulting graph distinguish between the attribute nodes and the primary-key nodes. (A primary-key node has one or more single-arrow links leaving it.) Mark the primary keys in some way (e.g. red color).

4. For each association between keys, add the inverse association if it is not already on the graph. If this results in an M:M link between keys, determine whether the inverse association would ever be used in reality. If it could be used at any time in the future replace it by introducing an extra concatenated key incorporating the key data items that were linked.

5. Examine the associations and identify any that appear redundant. For any associations that are candidates for

removal, check carefully that their meaning is genuinely redundant; if so, remove them.

6. Repeat the previous four steps until all user views are merged into the graph.

7. Identify the root keys. (A root key is a primary key with no single arrow leaving it to another key.)
For pictorial clarity the diagram should be rearranged with the root keys at the top. The single-arrow links between keys should point upward where possible. The links between primary keys may be marked in color.

8. Observe whether the graph contains any isolated attributes. An isolated attribute is a node with no single-arrow links entering or leaving it (only double-arrow links). An isolated attribute could be treated in one of three ways:

(a) It may be implemented as a repeating attribute in a variable-length record.
(b) It may be treated as a solitary key--a one-data-item record.
(c) It may be the result of an error in interpretation of the user's data, in which case the error is corrected.

9. Adjust the graph to avoid any intersecting attributes (an intersecting attribute with more than one single-arrow link entering it). An intersecting attribute can be avoided by:

(a) Replacing it with one or more links to it with equivalent links via existing key.
(b) Duplicating the data item group in question.
(c) Treating it as a solitary key--a one-data-item record.

10. Redraw the data items arranged into groups (records, segments, tuples), each having one primary key and its associated attributes. A group may now be drawn as a box. The boxes may be offset from the left to indicate their "depth" under the root group.

11. Identify all secondary key. (A secondary key is an attribute with one or more double-arrow links leaving it.) Draw the secondary-key links between the boxes.

12. To make the resulting model as stable as possible, apply the steps referred to in Chapter 17 on stability analysis.

13. The unconstrained "canonical" model may now be converted into the more constrained schema associated with a particular software package. It is generally a simple step to convert the canonical model into a CODASYL, DL/1, or relation schema. Some software, however, has constraint that would require a

major deviation from, or splitting of, the canonical view. Some software will simply not be able to handle it.

In converting the canonical model to a particular software schema, performance considerations associated with high-usage and fast-response paths should be examined. We suggest the following steps:

(a) Mark all paths which are used in interactive systems and which need fast response time.
(b) Estimate the number of times per month each user path will traversed. Add up how often each association will be traversed (in each direction when applicable).
(c) Estimate the length of each group.
(d) For each M associations, estimate the size of M; that is, how many values on average are associated with one value, or how many "child" groups are associated with a "parent" group.

The information above may affect the choice of structure and may cause the designer to modify the schema. In some cases a group may be split because it contains a mixture of frequently used and rarely used data, or is too long. In some cases a schema will be split to avoid complexity.

14. With the software schema designed, return to the original users views and ensure that they can be handled by it. In some cases the performance cost of handling a particular user view is sufficiently great that it is worthwhile completely modifying that user view."

Appendix: I

SADT Diagrams for UDB

Node Index

C1 A-0 Universal Data Base Management System
C2 A0 Evaluate UDB Transaction
C3 A1 Evaluate Queries
C4 A11 Analyze Query
A111 Is Query in LDML or UDML?
A112 Is Desired Information at least partially universal?
C7 A12 Modify Query
C8 A121 Translate to Universal DML
A1211 Translate IMS to UDML
A1212 Translate DBTG to UDM
A1213 Translate Sys. R to UDML
C12 A122 Parse and Optimize Query
C13 A123 Translate to Local DML
A1231 Translate Uni. to IMS DML
A1232 Translate Uni. to DBTG DML
A1233 Translate Uni. to Sys. R DML
C17 A2 Evaluate Data
C18 A21 Analyze Data
A211 Is Data in Local or Universal Format?
A212 Does Data Need to be in Local or Universal Format?
C21 A22 Modify Data
C22 A221 Translate Data to Local Format
A2211 Translate to IMS Data Format
A2212 Translate to DBTG Data Format
A2213 Translate to System R Data Format
C23 A222 Translate Data to Uni. format
A2221 Translate IMS Data to Universal Format
A2222 Translate DBTG Data to Universal Format
A2223 Translate System R Data to Universal Format

A-0 Universal Data Base Management System

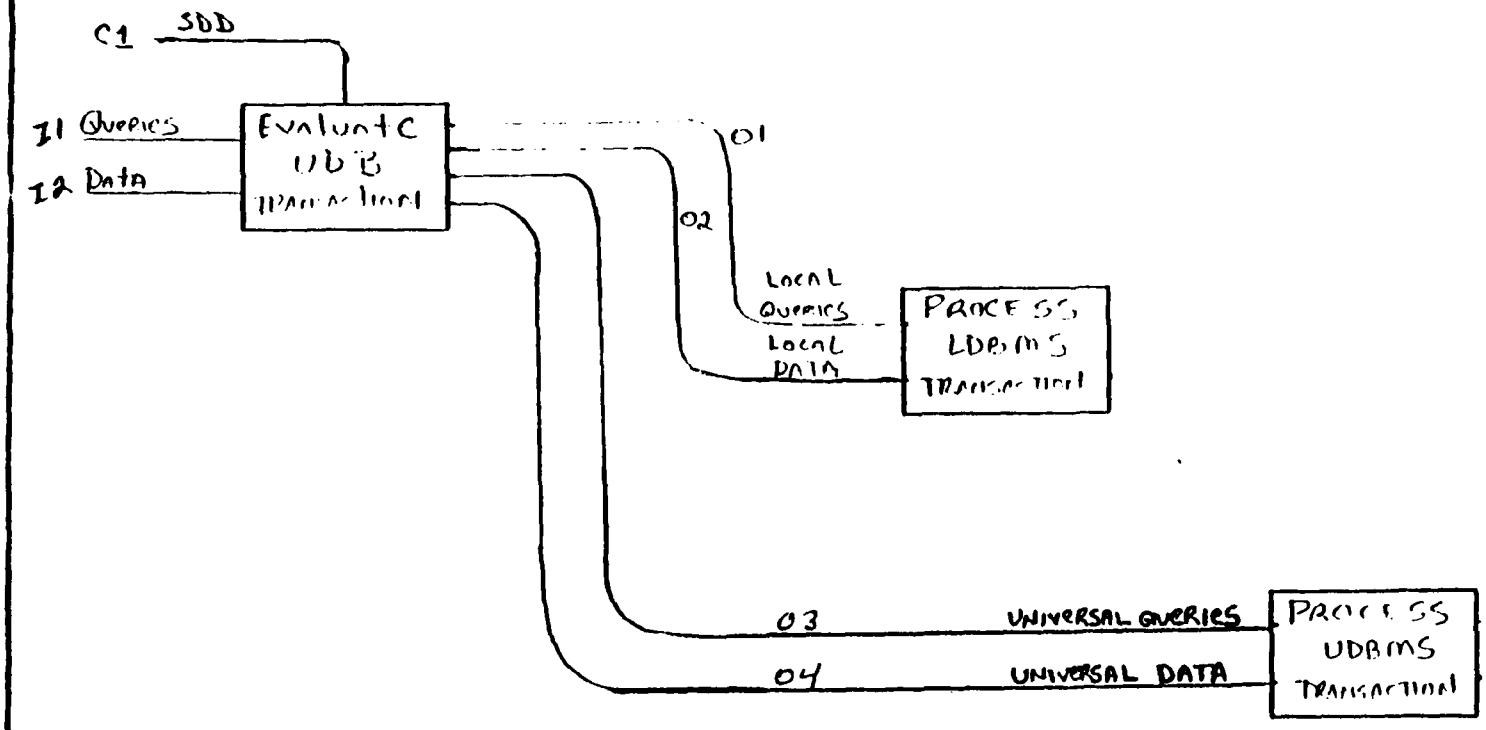
Abstract: This diagram indicates that queries and data enter (conceptually) the UDBMS and are broken down into local or universal queries and data. Local queries and data are routed to the local data base management system (LDBMS) while universally formatted queries and data are sent out on the network to the appropriate DBMS.

A-01 Evaluate UDB Transaction is visualized as existing at the local level only (each local DBMS (LDBMS) having its own UDBMS module) with no real global system actually existing in the sense of being one singular distinct entity. Basically, the UDBMS will be used as a front-end to the existing local DBMS. The LDBMS will func-tion as before and will not "realize" that the UDBMS exists, only that it is part of a distributed DBMS (DDBMS).

A-02 Process Local Data Base Management System (LDBMS) Transaction is not decomposed any further and merely represents the existence of a given LDBMS.

A-03: Process Universal Data Base Management System (UDBMS) Transaction is not decomposed any further and represents the existence of a DDBMS which is conceptually viewed as a singularly distinct UDBMS.

AUTHOR: 2LT Anthony J. Jones	DATE: 11 APR 87	READER				
PROJECT: UDBMS	REV:	DATE				



NODE: A-Ø	TITLE: UNIVERSAL DATA BASE MANAGEMENT SYSTEM	NUMBER: C1

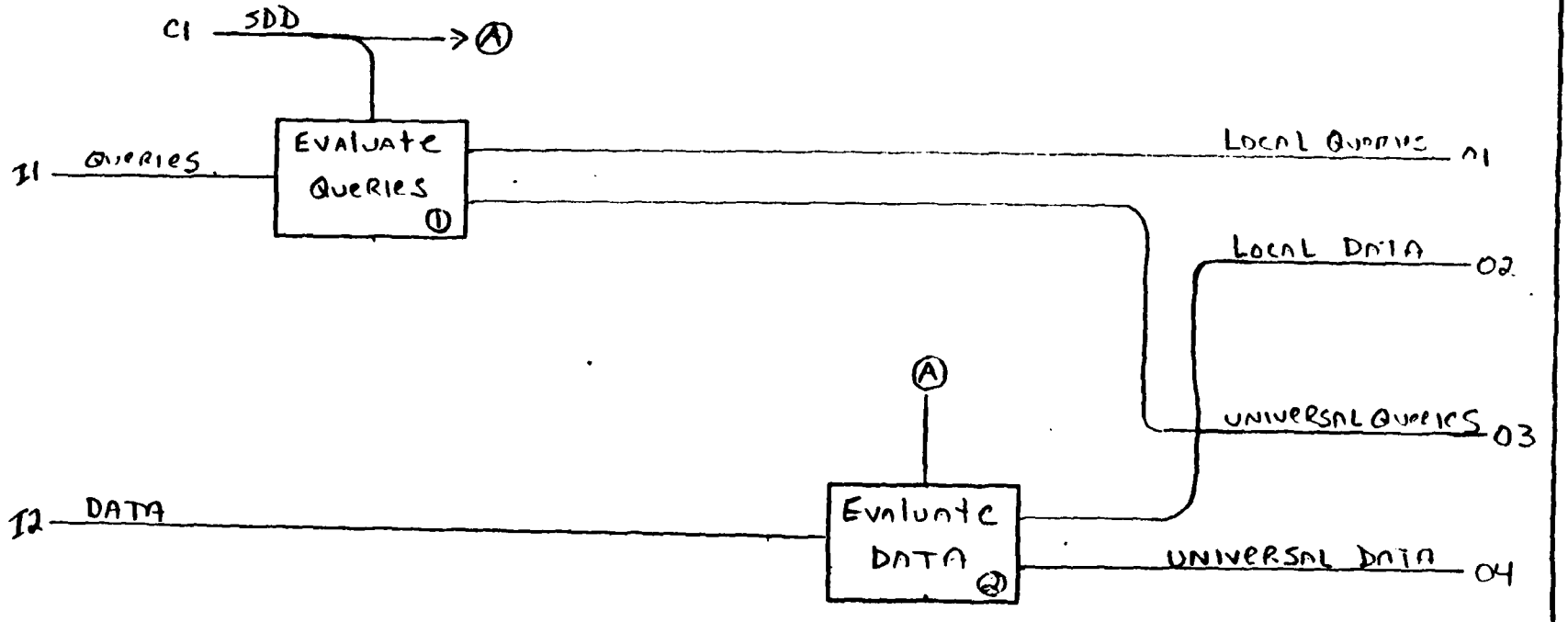
A0 Evaluate UDB Transaction

Abstract: This diagram indicates that queries and data enter the system and are handled separately. At this time all queries and data may be in either the local or universal language/data format. Local queries (and data) are queries that where submitted to the system by a local user of the system. Universal queries and data will be queries and data that origninated from another DBMS on the network. Queries and data from network users may arrive in the local language /data format if the DDBMS system happens to be the same model as the local one.

A1 Evaluate Queries takes queries in either the local or universal language, evaluates them, parses them appropriately, and routes the resulting subqueries to either the LDBMS or the correct GDBMSs (UDBMSs) for actual computation. Obviously, all queries entering the LDBMS from the network will involve information contained all locally. Local queries will be either all local, partially local, or all global. Local queries requiring global information will wait until that information is sent back to the LDBMS.

A2 Evaluate Data takes incoming or outgoing data packages, translates them into the necessary data format and routes them either to the LDBMS or some GDBMS on the network. Part of the data package will include a route message which indicates whether the package is incoming or outgoing.

AUTHOR: 2LT Anthony J. Jones	DATE: 11 Oct 84	READER				
PROJECT: UDBMS	REV:	DATE				



NODE: A0	TITLE: Evaluate UDB TRANSACTION	NUMBER: C2

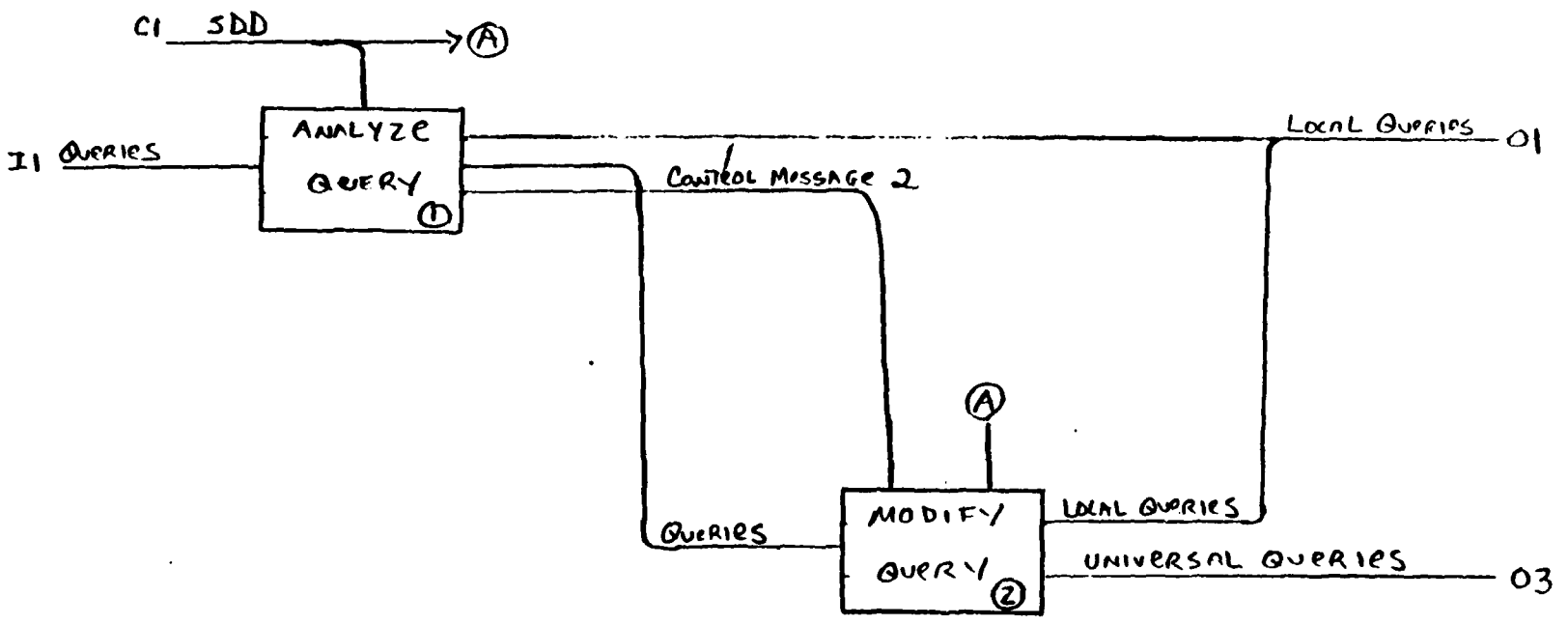
A1 Evaluate Queries

Abstract: Evaluate queries analyzes the query to determine if any translations are required.

A11 Analyze Query examines a query to determine if that query needs to be translated into the universal (or local) langauge. Queries in the local language, involving only locally stored information, are routed straight to the LDBMS.

A12 Modify Query translates a query from the universal to the local or the local to the universal language, optimizes the query, and then, if necessary, parses the query appropriately to gather any information on other DBMS.

AUTHOR: 2LT Anthony J. Jones	DATE: 11 Oct 84	READER				
PROJECT: UDBMS	REV:	DATE				

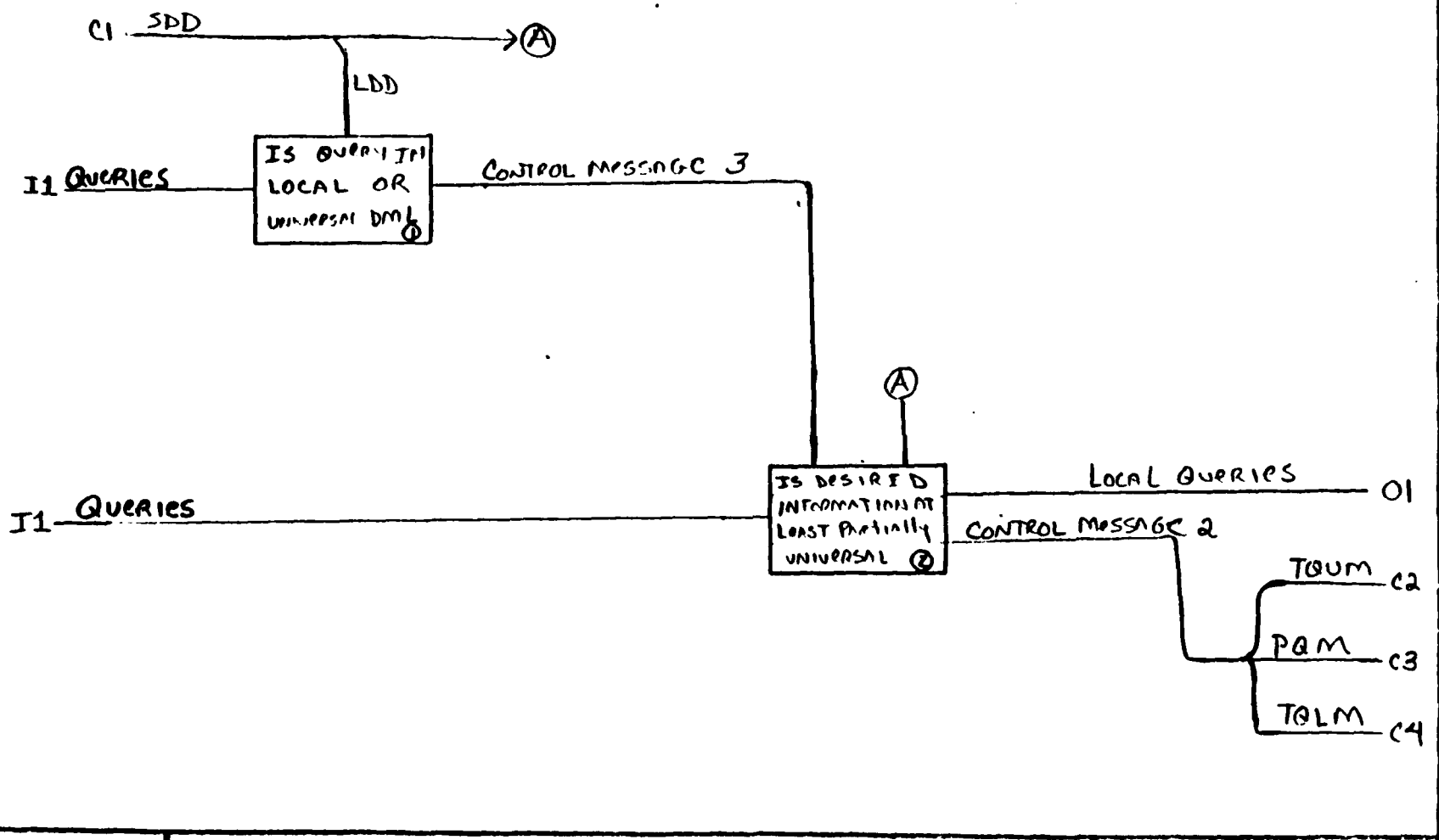


NODE: A1	TITLE: Evaluate Queries	NUMBER: C3

A11 Analyze Query

Abstract: This diagram shows how the analyzer evaluates a query. First it determines whether or not the query is in the universal or local language. Next, it determines where the information that the query desires is all local or at least partially global. If the query is in the local language and involves only local data then that query is routed straight to the LDBMS. Otherwise, control messages are generated for the query modfier (A12) to indicate to the query modifier how the query should be modified. The three messages generated are the PQM (Parse Query Message), the TQLM (Translate Query to Local Message), and the TQUM (Translate Query to Unversal Message - may require parsing).

AUTHOR: 2LT ANTHONY J. JONES	DATE: 11 Oct 84	READER				
PROJECT: UDBMS	REV:	DATE				

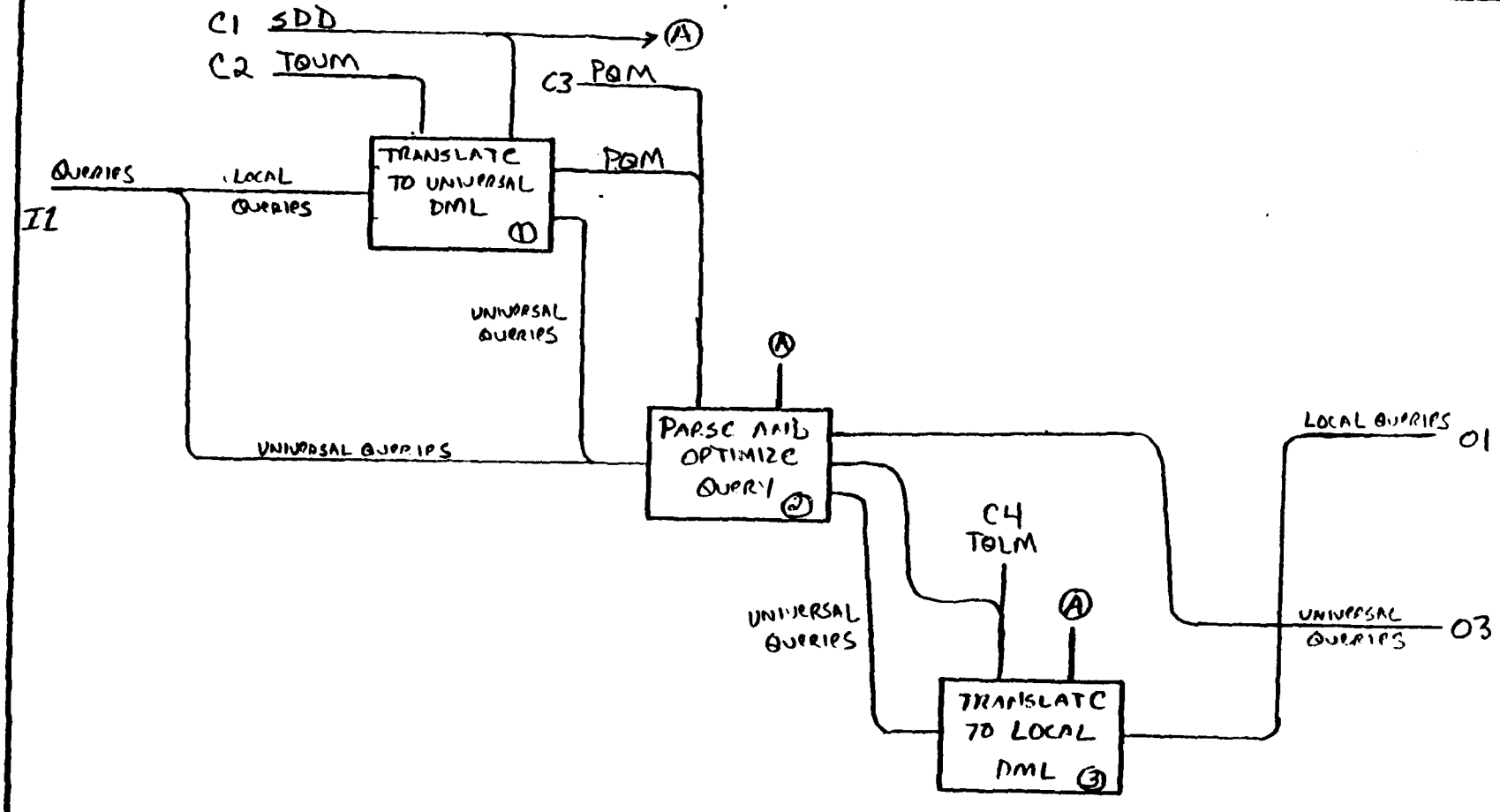


NODE:	TITLE:	NUMBER:
All	ANALYZE QUERY	C4

A12 Modify Query

Abstract: This diagram shows how the modify query routes queries to correct module for modificaton. Any query in the local language that needs at least a portion of its information from the UDBMS is translated into the universal langauge and sent to the parser and optimizer. It may end up that a subset of that universal query, originally in the local language, will be translated back into the local language and routed to the LDBMS. The LDBMS will be sent the original local query so that when all of the required data arrives from the network it can evaluate the original query. All queries in the universal langauge, which require only local information, are translated into the local langauge and routed to the LDBMS with a message indicating where to sent the results of the query.

AUTHOR: 2LT Anthony J. Joaks	DATE: 11 Oct 84	READER				
PROJECT: UDBMS	REV:	DATE				



NODE: A12	TITLE: MODIFY QUERY	NUMBER: C7

A121 Translate to Universal DML

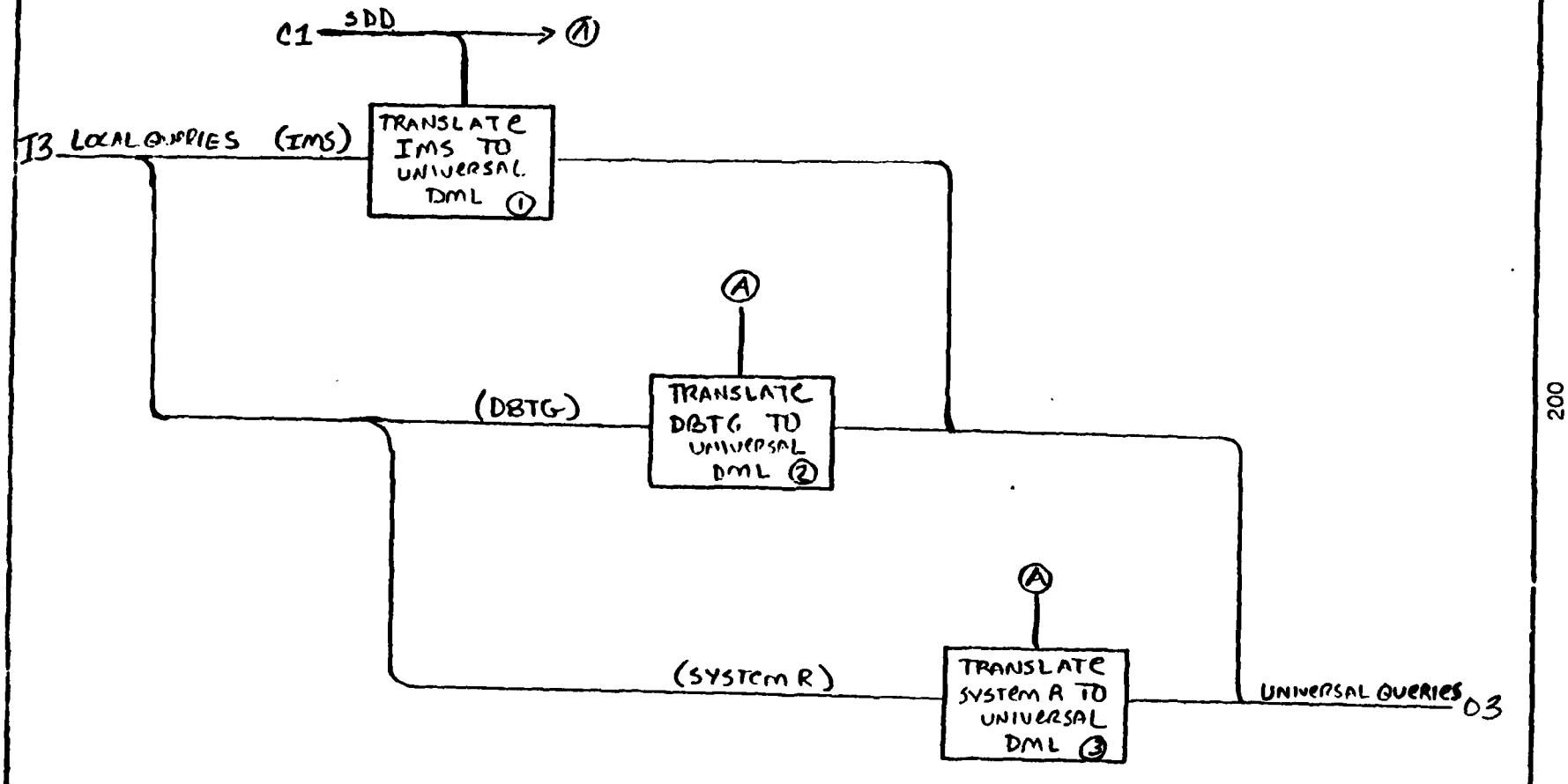
Abstract: This diagram shows how the LDMLs are translated to the UDML.

A1211 Translate IMS to Universal DML takes a query in the IMS DML and translates it to the UDML.

A1212 Translate DBTG to Universal DML takes a query in the DBTG DML and translates it to the UDML.

A1213 Translate System R to Universal DML takes a query in the System R DML and translates it to the UDML.

AUTHOR: 2LT Anthony J. Jones	DATE: 11 Oct 84	READER				
PROJECT: UDBMS	REV:	DATE				



NODE: A121	TITLE: TRANSLATE TO UNIVERSAL DML	NUMBER: C8

A123 Translate to Local DML

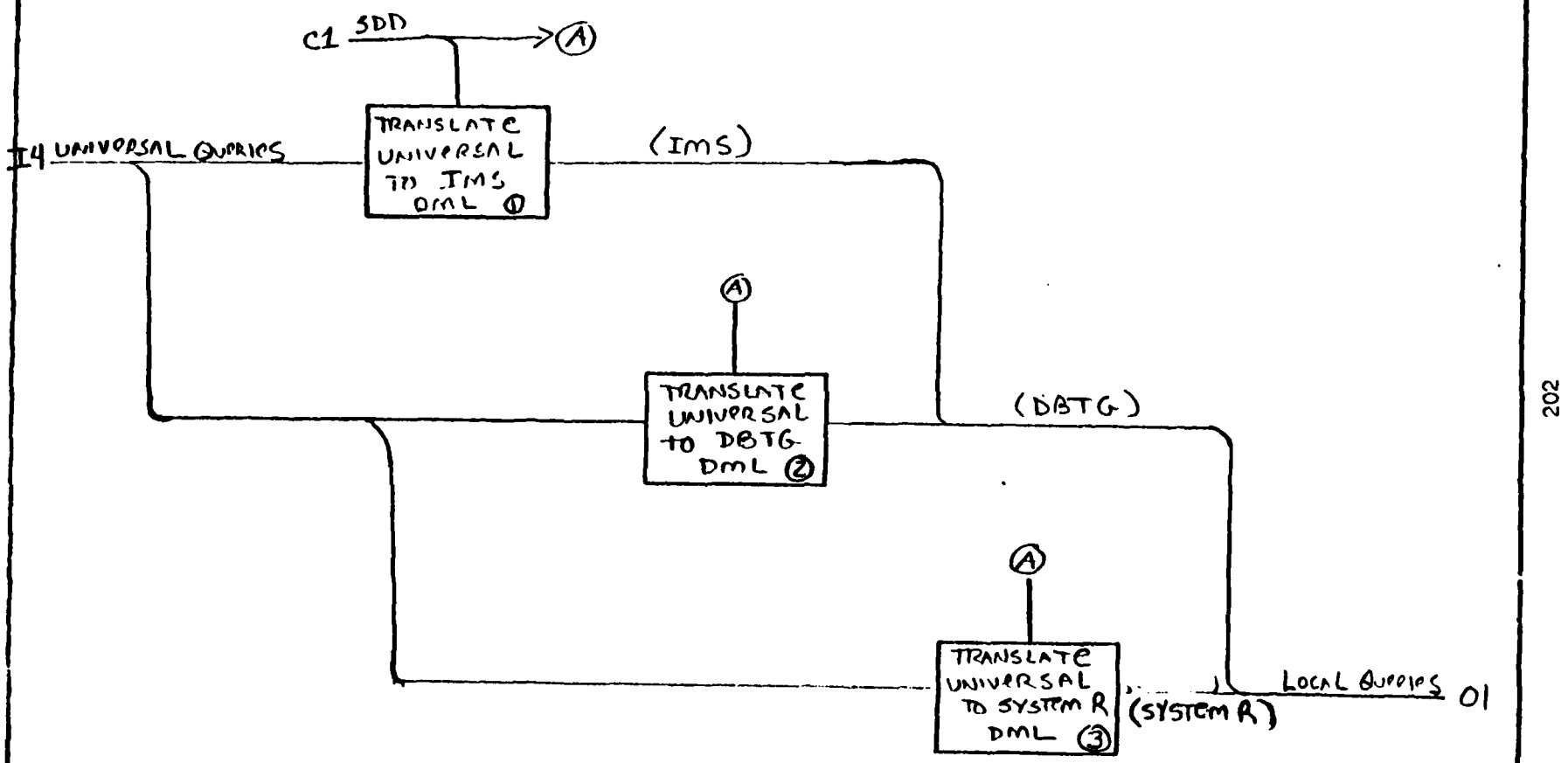
Abstract: This diagram shows how the UDMLs are translated to their respective LDMLs.

A1231 Translate Universal to IMS DML takes a query in the UDML and translates it to the IMS DML.

A1232 Translate Universal to DBTG DML takes a query in the UDML and translates it to the DBTG DML.

A1233 Translate Universal to System R DML takes a query in the UDML and translates it to the System R DML.

AUTHOR: 2LT Anthony J. Jones	DATE: 11 Oct 84	READER				
PROJECT: UDBMS	REV:	DATE				



NODE: A123	TITLE: TRANSLATE TO LOCAL DML	NUMBER: C13

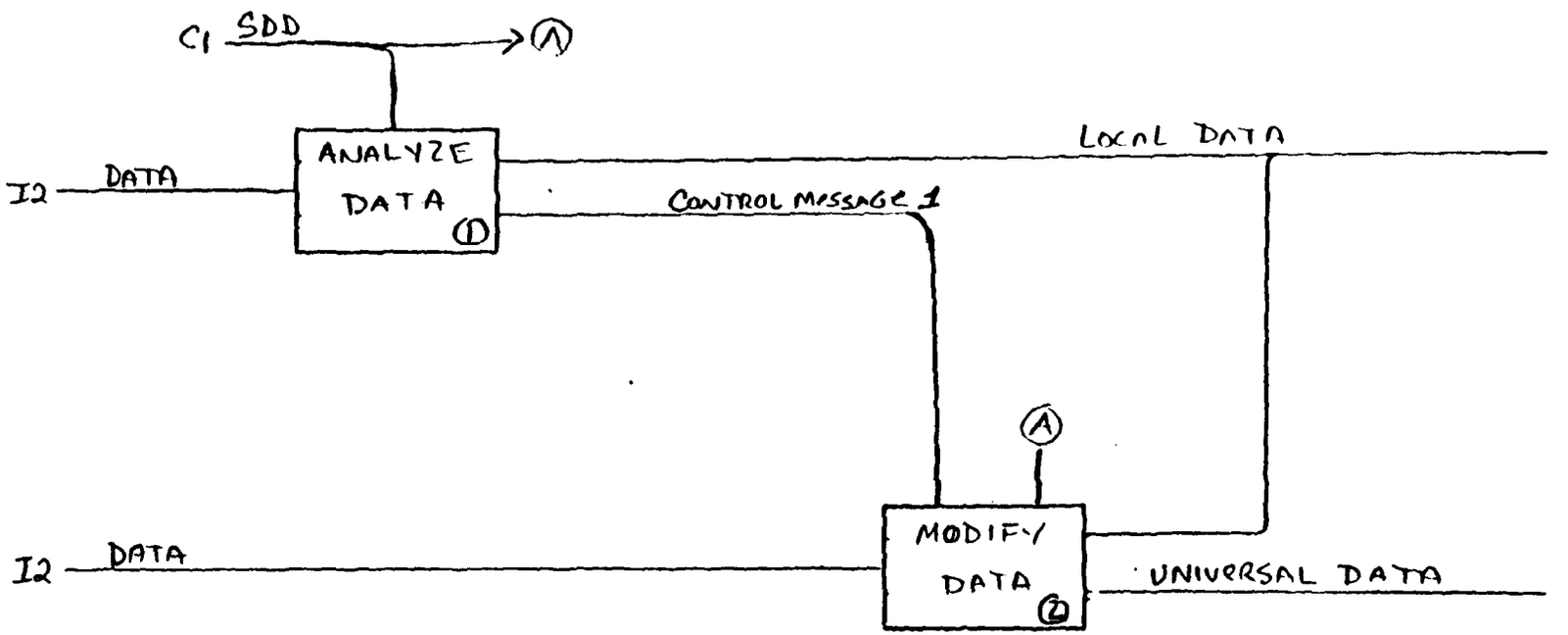
A2 Evaluate Data

Abstract: This diagram shows that the data is first analyzed to determine if and how it should be modified. If modification is required the data package is sent the the Data Modifier to accomplish this.

A21 Analyze Data determines if the data is in the local or universal format and determines what formats the information should be in for any requesting DBMS. Messages are generated for the Modify Data (A22) indicating what it should do.

A22: Modify Data translates the data packages from the local to the universal or the universal to the local.

AUTHOR: 2LT ANTHONY J. JACKS	DATE: 11 OCT 84	READER				
PROJECT: UDBMS	REV:	DATE				

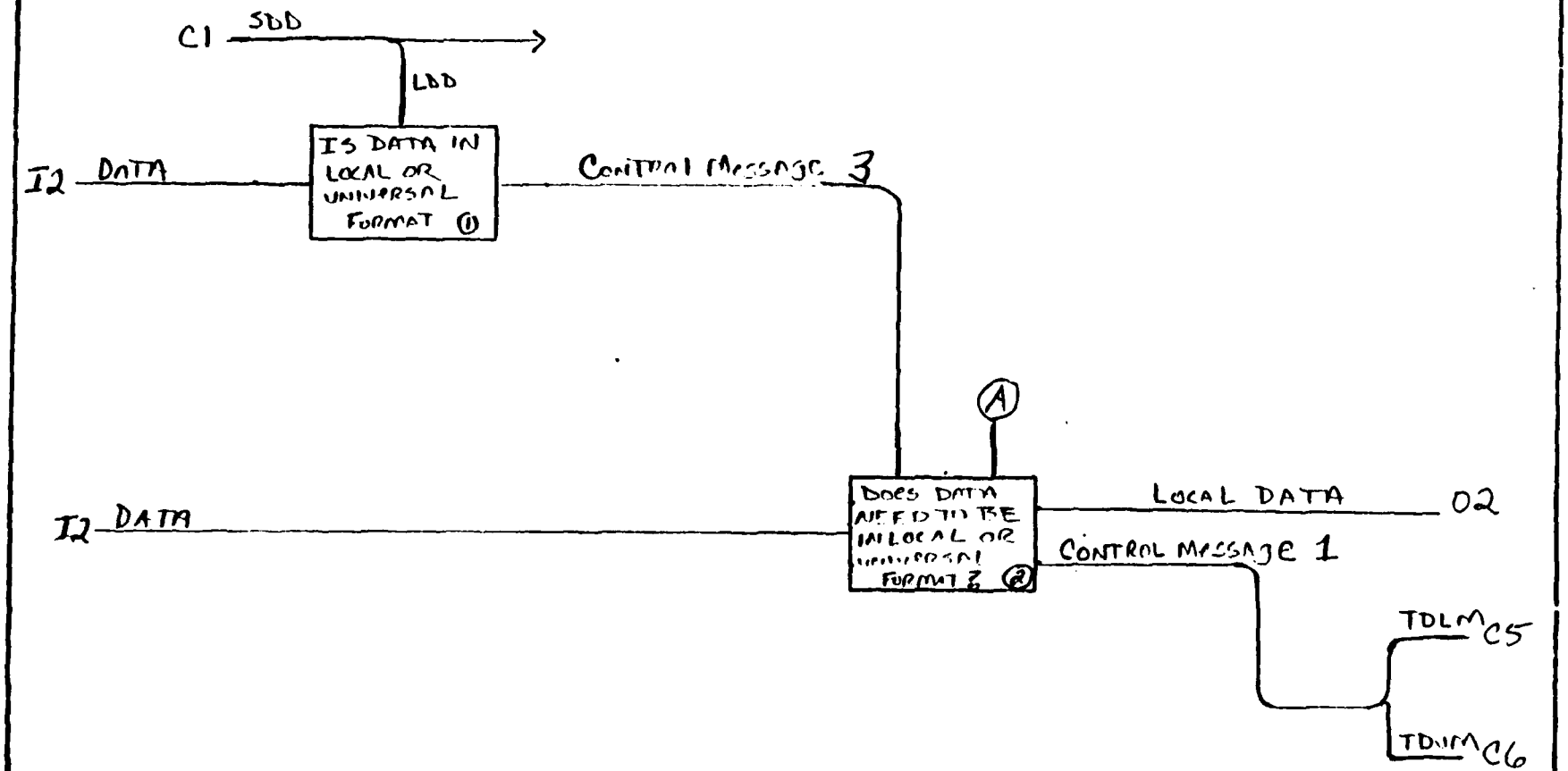


NODE: A2	TITLE: EVALUATE DATA	NUMBER: C17

A21 Analyze Data

Abstract: This diagram shows how the analyze data determines how and if a particular group of data should be translated from one data format (local/universal) to another (universal/local). If the data is in the particular format that is required then no translation is done. If the data is in the local format and needs to be in the universal, then a Translate Data to Universal Message (TDUM) is generated for the Modify Data module (A22) and the data is routed to it. If the data is in the universal format and needs to be in the local one, then a Translate Data to Local Message (TDLM) is generated for the Modify Data module and the data routed to it.

AUTHOR: 2LT Anthony J. Jones	DATE: 11 Oct 84	READER				
PROJECT: UDBMS	REV:	DATE				



206

NODE: A21	TITLE: ANALYZE DATA	NUMBER: C18

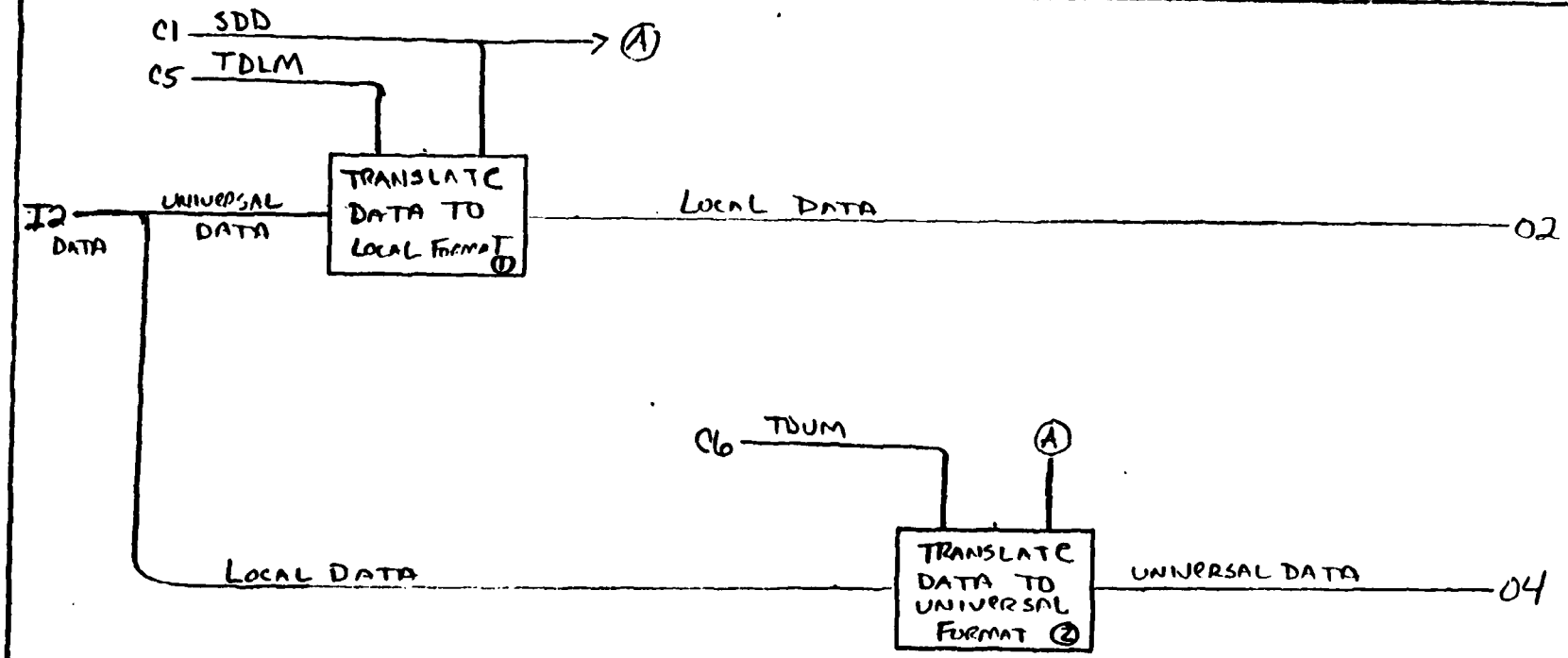
A22 Modify Data

Abstract: This diagram shows data coming into the Modify Data module and, depending on the control message generated by Analyze Data (A21), translates the data into the local or universal format. The System Data Dictionary contains the required information about each format to necessary to perform the translations.

A221 Translate Data to Local Format recieves data in universal format and translates it to the local data format.

A222 Translate Data to Universal Format recieves data in the local format and translates it to the universal format.

AUTHOR: 2LT Anthony J. Jones	DATE: 11 Oct 84	READER				
PROJECT: UDBMS	REV:	DATE				



NODE: A22	TITLE: MODIFY DATA	NUMBER: C21

A221 Translate Data to Local Format

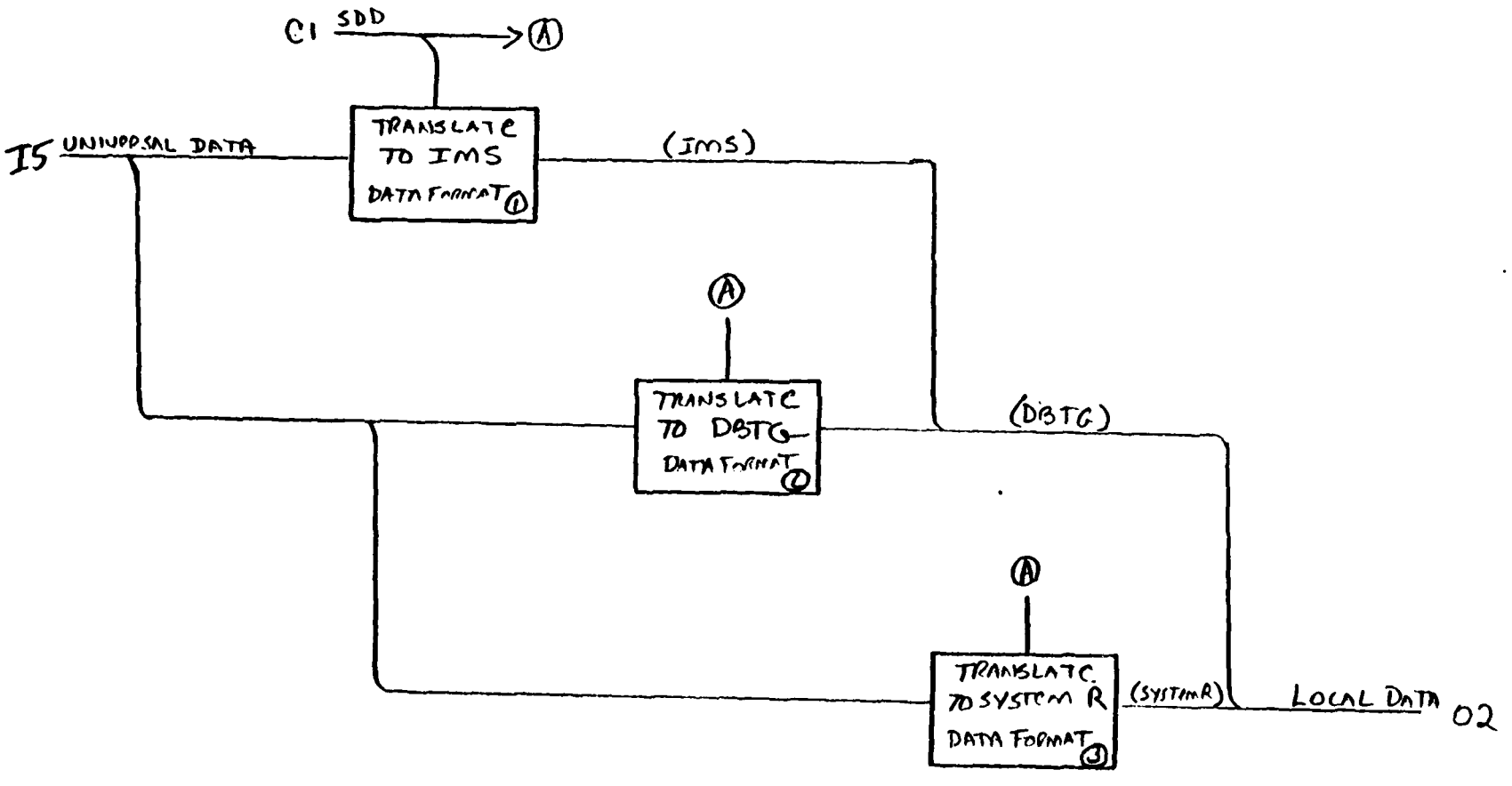
Abstract: This diagram shows that a given data package in the universal format is translated to the required local data format depending on which model the target data base is in.

A2211 Translate to IMS Data Format takes a universal data package and translates it to the IMS data format.

A2212 Translate to DBTG Data Format takes a universal data package and translates it to the DBTG data format.

A2213 Translate to System R Data Format takes a universal data package and translates it to the System R data format.

AUTHOR: 2LT Anthony J. Jones	DATE: 11 Oct 84	READER				
PROJECT: UDBMS	REV:	DATE				



NODE:	TITLE:	NUMBER:
A221	TRANSLATE DATA TO LOCAL FORMAT	C22

A222 Translate Data to Universal Format

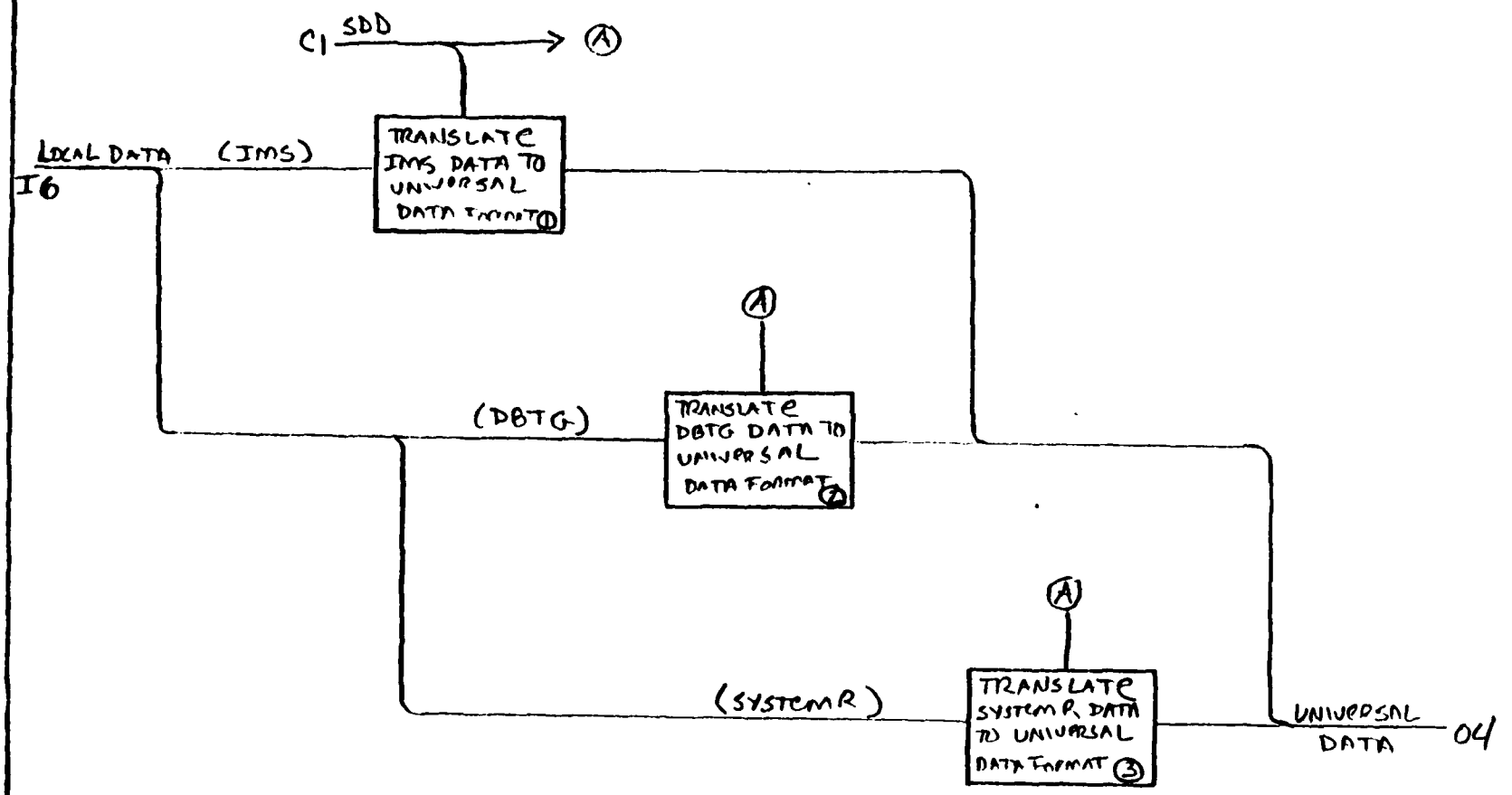
Abstract: This diagram shows that a given data package in a given local data format is translated to the universal data format.

A2221 Translate IMS Data to Universal Data Format takes a IMS data package and translates it to the universal data format.

A2222 Translate DBTG Data to Universal Data Format takes a DBTG data package and translates it to the universal data format.

A2223 Translate System R Data to Universal Format takes a System R data package and translates it to the universal data format.

AUTHOR: 2LT Anthony J. Jones	DATE: 11 Oct 84	READER				
PROJECT: UDBMS	REV:	DATE				



NODE: A222	TITLE: TRANSLATE DATA TO UNIVERSAL FORMAT	NUMBER: C23

Appendix: J

Data Dictionary Entries for UDB

Universal Data Base

Data Dictionary Entry for Activity

TYPE: ACTIVITY

DATE: 6 Oct 1984

NUMBER: A21

NAME: Analyze Data

INPUTS: Data

OUTPUTS: Control Message 1, Local Data

CONTROLS: System Data Dictionary

DESCRIPTION: Analyzes a data package to determine what, if any, modifications must be done. It controls whether or not the Modify Data activity is performed.

Universal Data Base

Data Dictionary Entry for Activity

TYPE: ACTIVITY

DATE: 6 Oct 1984

NUMBER: All

NAME: Analyze Query

INPUTS: Queries

OUTPUTS: Queries, Control Message 2, Local Queries

CONTROLS: System Data Dictionary

DESCRIPTION: Analyzes a query to determine what, if any, modifications must be done. It controls whether or not the Modify Query activity is performed.

Universal Data Base

Data Dictionary Entry for Data Element

TYPE: DATA ELEMENT

DATE: 6 Oct 84

NAME: Control Message 1

DESCRIPTION: Control message indicating how a data package should be translated.

SOURCES: A21

DESTINATIONS: A22

COMPOSITON: TDUM, TDLM

PART OF: N/A

DATA CHARACTERISTICS: N/A

VALUES: N/A

ALIASES: None

Universal Data Base

Data Dictionary Entry for Data Element

TYPE: DATA ELEMENT

DATE: 6 Oct 84

NAME: Control Message 2

DESCRIPTION: Control message indicating how a query should be translated.

SOURCES: A11

DESTINATIONS: A12

COMPOSITON: TQLM, TQUM, PQM

PART OF: N/A

DATA CHARACTERISTICS: N/A

VALUES: N/A

ALIASES: None

Universal Data Base

Data Dictionary Entry for Data Element

TYPE: DATA ELEMENT

DATE: 6 Oct 84

NAME: Control Message 3

DESCRIPTION: Message sent from one module to another indicating whether or not the query or data in question is in or needs to be in the universal or local form.

SOURCES: A111, A211

DESTINATIONS: A112, A212

COMPOSITON: N/A

PART OF: N/A

DATA CHARACTERISTICS: N/A

VALUES: N/A

ALIASES: None

Universal Data Base

Data Dictionary Entry for Data Element

TYPE: DATA ELEMENT

DATE: 6 Oct 84

NAME: Data

DESCRIPTION: Encompasses any type of data coming into the system or process. May be only composed of data in the local DBMS format, the universal data format, or both.

SOURCES: N/A

DESTINATIONS: A0, A2, A21, A22, A211, A212

COMPOSITON: Local Data, Universal Data

PART OF: N/A

DATA CHARACTERISTICS: Universal, IMS, CODASYL, or Relational data formats.

VALUES: N/A

ALIASES: None

Universal Data Base

Data Dictionary Entry for Activity

TYPE: ACTIVITY

DATE: 6 Oct 1984

NUMBER: A212

NAME: Does Data Need to be in the Local or Universal Format

INPUTS: Data

OUTPUTS: Local Data, Control Message 1

CONTROLS: System Data Dictionary, Control Message 3

DESCRIPTION: Self-explanatory. Answer to question used in conjunction with A211 to produce appropriate control message for A22. If the data needs to be in the local format then the TDLM message is generated otherwise the TDUM is generated.

Universal Data Base

Data Dictionary Entry for Activity

TYPE: ACTIVITY

DATE: 6 Oct 1984

NUMBER: A2

NAME: Evaluate Data

INPUTS: Data

OUTPUTS: Local Data, Universal Data

CONTROLS: System Data Dictionary

DESCRIPTION: Evaluates an incoming data package and determines if the package must be converted to the local format or the universal format.

Universal Data Base

Data Dictionary Entry for Activity

TYPE: ACTIVITY

DATE: 6 Oct 1984

NUMBER: A1

NAME: Evaluate Queries

INPUTS: Queries

OUTPUTS: Local Queries, Universal Queries

CONTROLS: System Data Dictionary

DESCRIPTION: Evaluates an incoming query, determines if any translations or parsing are required, performs them, and routes the resulting queries to the appropriate DBMSs.

Universal Data Base

Data Dictionary Entry for Activity

TYPE: ACTIVITY

DATE: 6 Oct 1984

NUMBER: AO

NAME: Evaluate UDB Transaction

INPUTS: Queries, Data

OUTPUTS: Local Queries, Local Data, Universal Queries, Universal Data

CONTROLS: System Data Dictionary

DESCRIPTION: Evaluates an incoming transaction which may be either a query or a data package. Each query or data package may originate from the UDBMS or LDBMS and could be in either the universal or local DML/data format. Queries or data in the local format could either originate from a system using the same model out on the network or from the LDBMS. Queries from the LDBMS are rejected if they involve universal data else they are simply routed to the LDBMS.

Universal Data Base

Data Dictionary Entry for Data Element

TYPE: DATA ELEMENT

DATE: 6 Oct 84

NAME: Global Data Dictionary (GDD)

DESCRIPTION: The GDD contains data dictionary entries and information about all of the data and data bases in the UDB system. The GDD will exist at a network site controlled by the UDBAC.

SOURCES: N/A

DESTINATIONS: N/A

COMPOSITON: N/A

PART OF: N/A

DATA CHARACTERISTICS: N/A

VALUES: N/A

ALIASES: None

Universal Data Base

Data Dictionary Entry for Activity

TYPE: ACTIVITY

DATE: 6 Oct 1984

NUMBER: A121

NAME: Translate to Universal DML

INPUTS: Local Queries

OUTPUTS: Universal Queries, PQM

CONTROLS: System Data Dictionary, TQUM

DESCRIPTION: Translates a LDML to the UDML.

Universal Data Base

Data Dictionary Entry for Activity

TYPE: ACTIVITY

DATE: 6 Oct 1984

NUMBER: A112

NAME: Is Desired Information at Least Partially Universal

INPUTS: Queries

OUTPUTS: Local Queries, Control Message 2

CONTROLS: System Data Dictionary, Control Message 3

DESCRIPTION: Self-explanatory. Answer to question used in conjunction with A111 to produce appropriate control message for A12. If the query is in the LDML and the desired information is only in the LDBMS, then the query is routed to the LDBMS. Otherwise the appropriate type of Control Message 2 is generated to indicate to A12 what to do. If the query is in the UDML and involves only universal information or local and universal, then the PQM message is sent (active) while the other two are not (inactive). If the query is in the UDML and involves only local information, then the TQLM is sent while the other two are not. If the query is in the LDML and involves global information the TQUM is sent while the other two are not. Note: This last ability may not be supported in the final UDB.

Universal Data Base

Data Dictionary Entry for Activity

TYPE: ACTIVITY

DATE: 6 Oct 1984

NUMBER: A111

NAME: Is Query in Local or Universal DML

INPUTS: Queries

OUTPUTS: Control Message 3

CONTROLS: Local Data Dictionary

DESCRIPTION: Self-explanatory. Answer to question used in conjunction with A112 to produce appropriate control message for A12.

Universal Data Base

Data Dictionary Entry for Data Element

TYPE: DATA ELEMENT

DATE: 6 Oct 84

NAME: Local Data

DESCRIPTION: Encompasses any type of data coming into the system or process which is formatted according to the LDBMS's data format.

SOURCES: A2, A22, A21, A212, A221, A2211, A2212, A2213

DESTINATIONS: A222, A2221, A2222, A2223

COMPOSITON: Data in local format.

PART OF: Data

DATA CHARACTERISTICS: IMS, CODASYL, or Relational data formats.

VALUES: N/A

ALIASES: None

Universal Data Base

Data Dictionary Entry for Data Element

TYPE: DATA ELEMENT

DATE: 6 Oct 84

NAME: Local Data Dictionary (LDD)

DESCRIPTION: The LDD contains data dictionary entries for all of the data contained within the local data base. It will contain the information necessary for evaluation and translation of queries and data packages.

SOURCES: N/A

DESTINATIONS: A111, A211

COMPOSITON: N/A

PART OF: System Data Dictionary (SDD)

DATA CHARACTERISTICS:

VALUES: N/A

ALIASES: None

Universal Data Base

Data Dictionary Entry for Data Element

TYPE: DATA ELEMENT

DATE: 6 Oct 84

NAME: Local Queries

DESCRIPTION: Encompasses any type of query coming into the system or process in the local DML.

SOURCES: A1, A11, A12, A112, A123, A1231, A1232, A1233

DESTINATIONS: A121, A1211, A1212, A1213

COMPOSITON: None

PART OF: Queries

DATA CHARACTERISTICS: IMS, CODASYL, or Relational DMLs.

VALUES: N/A

ALIASES: None

Universal Data Base

Data Dictionary Entry for Activity

TYPE: ACTIVITY

DATE: 6 Oct 1984

NUMBER: A22

NAME: Modify Data

INPUTS: Data

OUTPUTS: Universal Data, Local Data

CONTROLS: System Data Dictionary, Control Message 1

DESCRIPTION: According the information provided by the controls, modify query will translate data from the universal format to the local format.

Universal Data Base

Data Dictionary Entry for Activity

TYPE: ACTIVITY

DATE: 6 Oct 1984

NUMBER: A12

NAME: Modify Query

INPUTS: Queries

OUTPUTS: Universal Queries, Local Queries

CONTROLS: System Data Dictionary, Control Message 2

DESCRIPTION: According the information provided by the controls, modify query will translate queries from the UDML to the LDML and vice versa.

Universal Data Base

Data Dictionary Entry for Activity

TYPE: ACTIVITY

DATE: 6 Oct 1984

NUMBER: A122

NAME: Parse and Optimize Query

INPUTS: Universal Queries

OUTPUTS: Universal Queries, TQLM

CONTROLS: System Data Dictionary, PQM

DESCRIPTION: Takes a UDML query, parses it, optimizes it, routes any queries involving distributed information to the UDBMS and queries involving local information to A123 to be translated back into the LDML.

Universal Data Base

Data Dictionary Entry for Data Element

TYPE: DATA ELEMENT

DATE: 6 Oct 84

NAME: Parse Query Message (PQM)

DESCRIPTION: A control message indicating that the query in question, which is in the UDML, is ready to be parsed and optimized.

SOURCES: A121, A112

DESTINATIONS: A122

COMPOSITON: None

PART OF: Control Message 2

DATA CHARACTERISTICS: IMS, CODASYL, or Relational DMLs.

VALUES: N/A

ALIASES: None

Universal Data Base

Data Dictionary Entry for Data Element

TYPE: DATA ELEMENT

DATE: 6 Oct 84

NAME: Queries

DESCRIPTION: Encompasses any type of query (retieval, update, or deletion) coming into the system or process. May be only composed of local queries, universal queries, or both.

SOURCES: N/A

DESTINATIONS: A0, A1, A11, A12, A112, A111

COMPOSITON: Local Queries, Universal Queries

PART OF: None

DATA CHARACTERISTICS: Universal, IMS, CODASYL, or Relational DMLs.

VALUES: N/A

ALIASES: None

Universal Data Base

Data Dictionary Entry for Data Element

TYPE: DATA ELEMENT

DATE: 6 Oct 84

NAME: System Data Dictionary (SDD)

DESCRIPTION: The SDD represents the local data dictionary and the extended local data dictionary combined. The SDD indicates that both the LDD and ELDD may be required to evaluate whether or not a particular query or set of data is contained in the local system, the universal system, or both. They will also be used to determine what particular data format or DML a set of data or a query is in, and so forth.

SOURCES: N/A

DESTINATIONS: A0, A1, A11, A112, A12, A121, A1211, A1212, A1213, A122, A123, A1231, A1232, A1233, A2, A21, A211, A212, A22, A221, A2211, A2212, A2213, A222, A2221, A2222, A2223

COMPOSITON: Local Data Dictionary (LDD), Universal Data Dictionary (UDD)

PART OF: N/A

DATA CHARACTERISTICS: N/A

VALUES: N/A

ALIASES: None

Universal Data Base

Data Dictionary Entry for Activity

TYPE: ACTIVITY

DATE: 6 Oct 1984

NUMBER: A2222

NAME: Translate DBTG Data to Universal Format

INPUTS: Local Data

OUTPUTS: Universal Data Data

CONTROLS: System Data Dictionary

DESCRIPTION: Takes a data package in the DBTG format and translates it to the universal format.

Universal Data Base

Data Dictionary Entry for Activity

TYPE: ACTIVITY

DATE: 6 Oct 1984

NUMBER: A2221

NAME: Translate IMS Data to Universal Format

INPUTS: Local Data

OUTPUTS: Universal Data Data

CONTROLS: System Data Dictionary

DESCRIPTION: Takes a data package in the IMS format and translates it to the universal format.

Universal Data Base

Data Dictionary Entry for Activity

TYPE: ACTIVITY

DATE: 6 Oct 1984

NUMBER: A2223

NAME: Translate System R Data to Universal Format

INPUTS: Local Data

OUTPUTS: Universal Data Data

CONTROLS: System Data Dictionary

DESCRIPTION: Takes a data package in the System R format and translates it to the universal format.

Universal Data Base

Data Dictionary Entry for Activity

TYPE: ACTIVITY

DATE: 6 Oct 1984

NUMBER: A2121

NAME: Translate Data to Local Format

INPUTS: Universal Data

OUTPUTS: Local Data

CONTROLS: System Data Dictionary, TDLM

DESCRIPTION: Translates a data package in the universal format to the universal format.

Universal Data Base

Data Dictionary Entry for Activity

TYPE: ACTIVITY

DATE: 6 Oct 1984

NUMBER: A222

NAME: Translate Data to Universal Format

INPUTS: Local Data

OUTPUTS: Universal Data

CONTROLS: System Data Dictionary, TDUM

DESCRIPTION: Takes a data package in the local format and translates it to the universal format.

Universal Data Base

Data Dictionary Entry for Data Element

TYPE: DATA ELEMENT

DATE: 6 Oct 84

NAME: Translate Data to Local Model (TDLM)

DESCRIPTION: Control message indicating that a data package should be translated from the universal model/format to the local model/format.

SOURCES: A212

DESTINATIONS: A221

COMPOSITON: N/A

PART OF: Control Message 1

DATA CHARACTERISTICS: N/A

VALUES: N/A

ALIASES: None

Universal Data Base

Data Dictionary Entry for Data Element

TYPE: DATA ELEMENT

DATE: 6 Oct 84

NAME: Translate Data to Universal Model (TDUM)

DESCRIPTION: Control message indicating that a data package should be translated from the local model/format to the universal model/format.

SOURCES: A212

DESTINATIONS: A222

COMPOSITON: N/A

PART OF: Control Message 1

DATA CHARACTERISTICS: N/A

VALUES: N/A

ALIASES: None

Universal Data Base

Data Dictionary Entry for Activity

TYPE: ACTIVITY

DATE: 6 Oct 1984

NUMBER: A1212

NAME: Translate DBTG to Universal DML

INPUTS: Local (DBTG) Queries

OUTPUTS: Universal Queries

CONTROLS: System Data Dictionary

DESCRIPTION: Takes a LDML DBTG query and translates it to the UDML. Note: A given LDBMS may support more than one type of DBMS. All systems will not necessarily have A1231, A1232, and A1233. They will only have the ones they need.

Universal Data Base

Data Dictionary Entry for Activity

TYPE: ACTIVITY

DATE: 6 Oct 1984

NUMBER: A1211

NAME: Translate IMS to Universal DML

INPUTS: Local (IMS) Queries

OUTPUTS: Universal Queries

CONTROLS: System Data Dictionary

DESCRIPTION: Takes a LDML IMS query and translates it to the UDML. Note: A given LDBMS may support more than one type of DBMS. All systems will not necessarily have A1231, A1232, and A1233. They will only have the ones they need.

Universal Data Base

Data Dictionary Entry for Activity

TYPE: ACTIVITY

DATE: 6 Oct 1984

NUMBER: A1213

NAME: Translate System R to Universal DML

INPUTS: Local (System R) Queries

OUTPUTS: Universal Queries

CONTROLS: System Data Dictionary

DESCRIPTION: Takes a LDML System R query and translates it to the UDML. Note: A given LDBMS may support more than one type of DBMS. All systems will not necessarily have A1231, A1232, and A1233. They will only have the ones they need.

Universal Data Base

Data Dictionary Entry for Activity

TYPE: ACTIVITY

DATE: 6 Oct 1984

NUMBER: A2212

NAME: Translate to DBTG Data Format

INPUTS: Universal Data

OUTPUTS: Local Data

CONTROLS: System Data Dictionary

DESCRIPTION: Takes a universal data package and translates it to the DBTG data format.

Universal Data Base

Data Dictionary Entry for Activity

TYPE: ACTIVITY

DATE: 6 Oct 1984

NUMBER: A2211

NAME: Translate to IMS Data Format

INPUTS: Universal Data

OUTPUTS: Local Data

CONTROLS: System Data Dictionary

DESCRIPTION: Takes a universal data package and translates it to the IMS data format.

Universal Data Base

Data Dictionary Entry for Activity

TYPE: ACTIVITY

DATE: 6 Oct 1984

NUMBER: A2213

NAME: Translate to System R Data Format

INPUTS: Universal Data

OUTPUTS: Local Data

CONTROLS: System Data Dictionary

DESCRIPTION: Takes a universal data package and translates it to the System R data format.

Universal Data Base

Data Dictionary Entry for Activity

TYPE: ACTIVITY

DATE: 6 Oct 1984

NUMBER: A123

NAME: Translate to Local DML

INPUTS: Universal Queries

OUTPUTS: Local Queries

CONTROLS: System Data Dictionary, TQLM

DESCRIPTION: Takes a UDML query and translates it to the LDML.

Universal Data Base

Data Dictionary Entry for Activity

TYPE: ACTIVITY

DATE: 6 Oct 1984

NUMBER: A121

NAME: Translate to Universal DML

INPUTS: Local Queries

OUTPUTS: Universal Queries, PQM

CONTROLS: System Data Dictionary, TQUM

DESCRIPTION: Translates a LDML to the UDML.

Universal Data Base

Data Dictionary Entry for Data Element

TYPE: DATA ELEMENT

DATE: 6 Oct 84

NAME: Translate Query to Local Model (TQLM)

DESCRIPTION: Control message indicating that a query should be translated from the UDML to the LDML.

SOURCES: A112

DESTINATIONS: A123

COMPOSITON: N/A

PART OF: Control Message 2

DATA CHARACTERISTICS: N/A

VALUES: N/A

ALIASES: None

Universal Data Base

Data Dictionary Entry for Data Element

TYPE: DATA ELEMENT

DATE: 6 Oct 84

NAME: Translate Query to Universal Model (TQUM)

DESCRIPTION: Control message indicating that a query should be translated from a LDML to the UDML.

SOURCES: A112

DESTINATIONS: A121

COMPOSITON: N/A

PART OF: Control Message 2

DATA CHARACTERISTICS: N/A

VALUES: N/A

ALIASES: None

Universal Data Base

Data Dictionary Entry for Activity

TYPE: ACTIVITY

DATE: 6 Oct 1984

NUMBER: A1232

NAME: Translate Universal to DBTG DML

INPUTS: Universal Queries

OUTPUTS: Local (DBTG) Queries

CONTROLS: System Data Dictionary

DESCRIPTION: Takes a UDML query and translates it to the LDML. Note: A given LDBMS may support more than one type of DBMS. All systems will not necessarily have A1231, A1232, and A1233. They will only have the ones they need.

Universal Data Base

Data Dictionary Entry for Activity

TYPE: ACTIVITY

DATE: 6 Oct 1984

NUMBER: A1231

NAME: Translate Universal to IMS DML

INPUTS: Universal Queries

OUTPUTS: Local (IMS) Queries

CONTROLS: System Data Dictionary

DESCRIPTION: Takes a UDML query and translates it to the LDML. Note: A given LDBMS may support more than one type of DBMS. All systems will not necessarily have A1231, A1232, and A1233. They will only have the ones they need.

Universal Data Base

Data Dictionary Entry for Activity

TYPE: ACTIVITY

DATE: 6 Oct 1984

NUMBER: A1233

NAME: Translate Universal to System R DML

INPUTS: Universal Queries

OUTPUTS: Local (System R) Queries

CONTROLS: System Data Dictionary

DESCRIPTION: Takes a UDML query and translates it to the LDML. Note: A given LDBMS may support more than one type of DBMS. All systems will not necessarily have A1231, A1232, and A1233. They will only have the ones they need.

Universal Data Base

Data Dictionary Entry for Data Element

TYPE: DATA ELEMENT

DATE: 6 Oct 84

NAME: Universal Data

DESCRIPTION: Encompasses any type of data coming into the system or process which is formatted according to the UDBMS's data format.

SOURCES: A2, A22, A222, A2221, A2222, A2223

DESTINATIONS: A221, A2211, A2212, A2213

COMPOSITON: Data in UDBMS format.

PART OF: Data

DATA CHARACTERISTICS: UDBMS format.

VALUES: N/A

ALIASES: None

Universal Data Base

Data Dictionary Entry for Data Element

TYPE: DATA ELEMENT

DATE: 6 Oct 84

NAME: Universal Queries

DESCRIPTION: Encompasses any type of query coming into the system or process in the UDML.

SOURCES: A1, A12, A122, A1211, A1212, A1213

DESTINATIONS: A122, A1231, A1232, A1233, A123

COMPOSITON: N/A

PART OF: Queries

DATA CHARACTERISTICS: UDML.

VALUES: N/A

ALIASES: None

Appendix K: Summary Paper for

Analysis and Specification of A Universal Data Model

For Distributed Data Base Systems

ABSTRACT: The environment of a heterogeneous distributed data base system is examined. The primary goal being that of allowing for the effective communication between heterogeneous Data Base Management Systems (DBMS) in a distributed environment. This communication is accomplished through an intermediate mapping model, named the Universal Data Model (UDM). Three models, the Canonical, Entity-Relationship, and the Relational, are comparatively evaluated as possible candidates and the relational model chosen as the UDM. Examples of the DML and DDL developed for the UDB are given. Several Data Model mapping issues are discussed.

Introduction

Since the inception of Data Base Management Systems (DBMS), their importance and impact on data processing has grown rapidly. One of the problems facing DBMSs and their users is the fact that while it would be very useful to allow distinct data bases to communicate with each other, the different data models used by each makes this very difficult. These different data models dictate how the data within the data bases is structured and manipulated.

The goal of this thesis was to develop a "Universal Data Base" (UDB) through which these distributed heterogeneous data bases could communicate and exchange/share information. The UDB is, of course, primarily logical in nature and presents its users with a universal representation of certain subset of information (for which they are interested in) in

which the actual physical location and structure of the data is unimportant. The UDB, besides being a concept or an environment, is comprised of a Universal Data Model (UDM), Universal Data Definition Language (UDDL), and Universal Data Manipulation Language (UDML). After examining the UDB as a whole, this thesis focused on the UDM and related mappings needed.

Overview

The thesis effort being summarized in this paper sought to permit heterogeneous DBMSs to communicate in a distributed environment. The thesis effort primarily focused on selecting a UDM and examining the necessary mappings between the local data base models and the UDM. However, to accomplish the selection of the UDM, a system requirements analysis was necessary for the UDB as a whole. Therefore, much of the thesis's effort involved this analysis. In this summary paper, the high points of the system's analysis will be discussed with the different candidates and selection criteria for the UDM briefly described. These criteria are then applied and a UDM choice specified. Sample DDL and DML commands are shown and the more important DDL mapping issues briefly discussed. This paper concludes with a summary of what was accomplished and what was not.

System's Analysis

The UDB will function in an extremely complex environ-

ment with different data bases, different data models, different users, and so forth. There are two major factors which will most influence the design of the UDB. These are: (1) the environment in which the UDB will function, and (2) the user's view of the UDB. The highlights of both of these factors are discussed in the following two subsections.

Environment. The major observations or conclusions of the impact of the environment on the UDB are described below:

> Observation 1: The current environment consists of independent, heterogeneous (or homogeneous) systems which have already established data bases, application programs and procedures.

The heterogeneous nature of this environment applies not only to the data base management systems that exist but also the host computer systems. Further, heterogeneous, in the DBMS context, not only indicates differences in the model used but also the particular implementation of any given model (i. e. two DBMSs using the same model could be different because the model was implemented in a different way in each model).

> Observation 2: Given that the owners of these systems will be very reluctant, at least all at once, to replace their older systems with any new systems, rewrite their application programs and/or retrain their people, the UDB will have to function in such a way as to minimize any of the aforementioned activities as much as possible.

Obviously, from a requirements standpoint, it would be optimal to have the situation where the user could simply be

informed that his system has been expanded to include more information and other than that, nothing has changed. From the design view, it would be very desirable to make everyone switch over completely to a single new system. Neither of the above viewpoints prove to be practical ones. The design point of view is impractical for cost reasons; and the requirements point of view because it presents some very tough problems. The first of these problems is that if the user is to view all the data as in his local system, then how will the data actually outside his system be presented to him? In what form will this data be presented? His local model may not be able to express the structures and constraints of another model effectively. Should a new, unrelated model be used to present all of the global data to the user? If so, which model should be chosen? This will, of course, force the user to learn the new model. If this model proves so flexible, why not use it instead of the local one altogether? Furthermore, what language will the local users work with? Will they work in their local language, which would be translated into the universal language, or will they have to work in the universal language? Finally, it should be noted that the independent local systems might actually reside on the same or different physical location on he same machine.

Observation 3: The UDB will function in an environment in which the users are reasonably cooperative and non-threatening to each other.

One of the attractive properties of a universal system, besides merely increasing the amount of available information, is that if certain subsets of information, e. g. personnel information, seemed particularly suited to a particular model, e. g. the relational, then all of the personnel information of the independent systems could be moved to a different data base system on the network. This is certainly one of the more extreme benefits, or possible uses for the UDB, but it does illustrate its potential. However, this type of utilization, as well as any less elaborate use, requires a cooperative and non-threatening environment. Even simply allowing users access to other user's information requires cooperation and a "non-threatening" environment. Data security and integrity present problems. Users, perhaps competitors, could seek to illegally access or modify informamation of other users. At present, data base security methods do not provide cost and performance effective protection.

Observation 4: The UDB will best function within a large governmental, military, or civilian organization.

Unfortunately, at this time, it proves rather impractical to have the UDB tying together all of the data bases in the "world". However, this does not really detract from the value of the UDB. For the most part, it is large organizations that have the most need for the UDB, and the greatest potential gain. Eventually, perhaps a nationwide or world-

wide UDB setup might be possible but not at this time. The complex issues of data security prove too difficult.

> Observation 5: The prime objective of the UDB is to develop a model to allow the prominent three models to interact.

The primary objective is not to develop a new superior model. The development of a new superior model, while certainly beneficial, would not necessarily solve the problem of networking the existing data base systems together. As noted earlier, organizations are not going to want to give up their present systems and investment, even for some new superior model. If the universal model proves superior then the various organizations may eventually convert. However, this will hopefully not be a requirement for becoming part of a UDB system.

> Observation 6: There are particular instances where each one of the three present models proves to be the best model for that instance. Their elimination from use may, therefore, not be optimal unless the new model proves able to recognize all of the strengths of the three models without their limitations.

Assuming that the new universal model does not prove to be a superior model, having the different models will prove useful to the users of the UDB. It is certainly true that there exist data base instances wherein each particular model will prove to be the best for that instance. Therefore, a user of the UDB could have, for example, several relational and network data bases, and a few heirarchical. When a user

decides to establish a data base, the user or UDBAC can choose which one of the three models it would best fit. The user, who may be used to working on relational systems, will, in some fashion, deal with the new data base in either the relational model or the universal model.

The User's View. An important consideration is how the user will view the information in the UDB and how he/she will interact with that information. Specifically, will the user view the information within the UDB through his local model and, therefore, use the local DML for queries or will the user use the universal model and DML? It was decided that because of the complexity in allowing the user to work within his local model, he would be forced to work through the universal model and DML.

The Approach

There were several different possible approaches to solving the requirements of the UDB. The approach taken by the thesis in question was to use an intermediate data model in which the local models would mapped into and the query processed through the universal model. It was decided, due to the complexities of handling all of the variants of each of the prominent three data models, that only one particular version of each of the three would be considered as valid local models. These three were: IMS for the heirarchical model, DBTG for the network model, and the System R for the relational model. It was further decided, that for now,

users would be restricted to using the universal model/ language for all universal queries (i. e. queries involving information at more than one sight, a global query).

The UDM Candidates

Three models were chosen as possible candidates for the UDM. These three models were chosen as representitives of the various classes of models that now exist. The classification used was based on how close the given model was to a real application. The Canonical model by Martin represents the totally logical model extreme. The Relational model represents the opposite end where the model is actually used for DBMS applications. The Entity-Relationship model by Chen represents the middle ground. It is a logical model, but is heavily based on the three "real" models, the network, heirarchical, and relational.

The Canonical Model. The Canonical model by Martin is defined as:

> ". . . a model of data which represents the inherent structure of that data and hence is independent of individual applications of the data and also of the software or hardware mechanisms that are employed in representing and using the data (10:235)."

The Canonical model was primarily developed to aid in designing data bases and is as much a process as a model. The model uses bubble charts and is a product of a process called "Canonical Synthesis". Due to its incremental nature, the canonical process (model and synthesis) proves to be particularly good at handling dynamic data base instances.

The synthesis process is readily automated, although still requiring some amount of human supervision and modification for total effectiveness. The resulting canonical model derived from the synthesis is independent of any model, performance constraints, or particular machine, and is in third normal form.

> "A canonical database structure is a minimal non-redundant model. Its records are in third normal form (and fourth normal form). . . [and] is sometimes referred to as a 'conceptual schema'. . . A canonical model can be represented as a network (CODASYL), heirarchical (IMS), or relational database system. A large canonical model may be kept, updated, and designed by computer, to represent overall the data which are the foundation of a computerized enterprise (10:275-276)."

Entity-Relationship Model. Entity-Relationship (ER) models are based on tables and graphs and were an outgrowth of the designing of data bases using commercial DBMS. Due to this fact, ER models bear a strong resemblance to the heirarchical and network models.

The ER model discussed was proposed by Chen (2) in 1976 and is considered to be probably the best known of the ER models (9:175). Originally, Chen's ER model was designed for the purpose of data base design by allowing the specification of an enterprise schema. An enterprise schema represents an enterprise's view of its data, independent of storage or efficiency considerations. Unlike the other ER models, Chen's ER model's conceptual schema is not necessarily directly accessible by a DBMS. The ER model only documents the logical properties of the data base and may or may not be directly

accessible.

The Relational Model. The relational model was originally proposed by Codd and arose out of a desire to bring some sort of formalism in addressing various issues and problems in the area of data base design. The obvious answer was to use already formulated mathematical theory. The following three observations/definitions provide a good idea of what the relational model is.

> "The relational approach to data is based on the realization that files that obey certain constraints may be considered as mathematical relations, and hence that elementary relation theory may be brought to bear on various practical problems of dealing with data in such files (3:65)."

> "Definition: Given a collection of sets D1, D2, . . ., DN (not necessarily distinct), R is a relation on those n sets if it is a set of ordered n-tuples <d1, d2, . . ., dn> such that d1 belongs to D1, d2 belongs to D2, . . ., dn belongs to DN. Sets D1, D2, D3 are the domains of R. The value n is the degree of R (3:83)."

> "The relational model, as defined by Codd [COD82] consists of three basic parts, a collection of relations that describe the logical structure of the database, a collection of operators to manipulate data stored in the database, and a collection of general integrity rules that constrain the set of valid states of the database (8:47-48)."

The relational model is a very simple and powerful model which has proved to be very popular in recent years.

The Universal Data Model

In this section, each of the three candidates is described in terms of each model's strengths and weaknesses in

the universal environment.

Relational Model. The relational model possesses a strong capability in a UDB application. The relational model is a proven model, its capabilities are known (although debated). The necessary mapping algorithms to the other two models have already been developed, or at least examined. The relational model is very simple and easy to learn and use. A summary of the relational model's strengths are listed below:

RS1. Simple and easy to learn and use.

The key to this strength is the fact that the relational model is not only easy to use but also easy to learn. The latter factor will prove especially useful if it is decided that the users will have to work in the universal model and language.

RS2. Table format lends itself well to a distributed environment.

This is, perhaps, the most important strength for the relational model. The table format of the relational model lends itself particularly well to the UDB application. It aides in presenting unrelated information and additional information since these can be handled simply by the addition of more tables (relations).

RS3. A well established model. Its capabilities are known as well as its weaknesses. Many of the mappings (relational to DBTG, etc.) have been done or at least examined.

Obviously, having an established model will make the implementation and acceptance of this model somewhat easier. There would be some savings in terms of the fact that the relational to relational mappings would be easier and the relational users would have to make a minimal adjustment. Introducing a relatively unknown model would involve more work and research then with this model.

RS4. The relational model shields the user from the underlying data formats and complexity of the data structures.

The primary value of this strength, in the UDB application, is the reduction in complexity.

RS5. It has a high level, nonprocedural DML which has proved to be easier to use and more productive for programmers (6:4).

Although the it would be desirable for the UDML to support both procedural and nonprocedural operations, it is important to note that it may not be practical nor preferably to support both. Nonprocedural languages are generally easier to use and more productive for the programmers. Furthermore, it may not be possible to support navigational operations in a distributed environment.

RS6. Storage and data structures are very simple (6:4).

A minor strength in this particular application.

RS7. Access paths don't have to be predefined (contrary to procedural languages/models) (6:4).

This is a particularly good feature of the relational model. Besides increasing the flexibility and power of the

model, this feature will prove extremely beneficial in a distributed environment.

RS8. The relational model has a fast response time to ad hoc queries which are considered to be a high-percentage of the queries submitted (6:5).

Certainly a strength for a regular DBMS, and still to some extent the UDBMS, but its impact in the UDBMS will not be very significant since the model would only be functioning as a communication media. It would not be actually processing queries (the LDBMSs would be).

RS9. The relational model handles M:M relationships extremely well.

M:M relationships can prove difficult to implement. The relational model handles this situation quite well because the M:M relationship, in terms of the data structures, is purely logical and is not physically set up.

RS10. The relational DML is highly parseable and well suited to optimization.

This is another strength which will be very valuable in the UDB application due to the distributed nature of the application.

The weaknesses of the relational model are summarized below:

RW1. The relational model is one which has been implemented and, therefore, has taken into consideration machine efficiency, etc. Perhaps, since the desired model need only appear to be a real DBMS, a purely logical model (canonical or ER) might be more flexible or better suited to the UDM role.

RW2. Even though the relational model (and operations) can be mapped to the network and heirarchical models (nonprocedural to procedural), perhaps it

would be more efficient to have a model with the built-in ability to do procedural operations rather than just being mapped into them.

RW3. The relational model cannot convey procedural operations.

If the relational model is chosen as the UDM, universal procedural operations will not be supported. Obviously, procedural operations could be mapped into the relational model but this would done taking into consideration the entire query. Essentially, a user could not single step through a distributed data base without extreme difficulty in most cases and almost impossible in others.

RW3. In general, perhaps a better approach would be to tailor the UDM to its environment rather than trying to fit the relational model into that role.

RW4. All constraints are not explicit.

Selection Criteria

This section describes the set of criteria used in comparing the three models.

Criteria #1: Simplicity and User Friendliness.

Criteria #2: Ability to depict all three models.

This particular criteria indicates how well the model can present the data base in the three different models. It will determine if the users will work in their local or the universal model.

Criteria #3: Ability to handle nonprocedural operations.

Criteria #4: Ability to handle procedural operations.

Criteria #5: Implementation benefits.

This criteria evaluates how much of a benefit will be derived, in terms of implementation, from choosing this model. This criteria is best shown by the relational model which has already been implemented with many of the "bugs" worked out.

Criteria #6: Ability to function in distributed environment.

This criteria evaluates how well the model handles the problems of distributed information being represented and manipulated.

Criteria #7: Ability to represent different relationships.

This criteria centers on how well the model represents the different relationships (1:1, 1:M, M:M).

Criteria #8: Flexibility in specification of relationships.

This criteria centers on how whether or not the model has predefined or non-predefined relationships.

Criteria #9: Ability to incorporate different user views.

Criteria #10: Ability to support all DBMS functions.

Criteria #11: Ability to express constraints.

This criteria evaluates how explicit the constraints are in each model and how many different constraints can be expressed.

Application of Criteria

In this section the criteria just described are weighted and applied against each of the UDM candidates. The models are rated comparatively for each criteria (see Tables I and II). The model which best satisfies that criteria receives a

three, the second best a two, and third best a one. If two models tie, they are both given the same score. Each criteria is weighted on a scale of one to five with five being the most weight. The criteria are weighted according to which are more important. The more important the criteria, the more weight it is given.

The Universal Model

An examination of Table II brings about the unfortunate observation that the relational model and ER model have tied for the honor of being the UDM. Obviously, only one model can be used for this thesis. Therefore, which one? Both the relational and the ER models offer certain advantages although they both come out equal in overall terms. To break this tie, criteria #3 is removed from consideration. The justification for this is the fact that the support of procedural operations in the distributed environment is impractical to implement. This brings the point totals to 67 (relational), 61 (ER), and 46 (canonical). Therefore, the relational model is chosen as the UDM.

Table I. Summary of Criteria

1. Simplicity and User Friendliness
2. Ability to depict all three models.
3. Ability to handle nonprocedural operations.
4. Ability to handle procedural operations.
5. Implementation benefits.
6. Ability to function in distributed environment.
7. Ability to represent different relationships.
8. Flexibility in specification of relationships.
9. Ability to incorporate different views.
10. Ability to support all DBMS functions.
11. Ability to express constraints.

Table II. Comparative Evaluation of Three UDM Candidates

Criteria	Weight	Relational	ER	Canonical
1	2	3	2	2
2	4	1	3	2
3	3	3	2	1
4	3	1	3	3
5	1	3	2	2
6	5	3	2	1
7	2	3	3	3
8	3	3	2	1
9	3	2	1	3
10	1	3	3	3
11	3	2	3	1
Total	30	70	70	55

Sensitivity Analysis

The purpose of this section is to investigate how sensitive the results of the evaluation weighting is to change. The approach to this analysis is to choose the two most significant (#2 and #6) criteria and evaluate the resulting change in the final totals from changing the weight factor in each up (a) or down (b) by one. This will be done with all eleven criteria (Start) and with criteria #4 removed (Minus). The table below depicts the resulting changes:

AD-A151 856 ANALYSIS AND SPECIFICATION OF A UNIVERSAL DATA MODEL FOR DISTRIBUTED DATA..(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. A J JONES

4/4

UNCLASSIFIED 14 DEC 84 AFIT/GCS/ENG/84D-11 F/G 9/2 NL

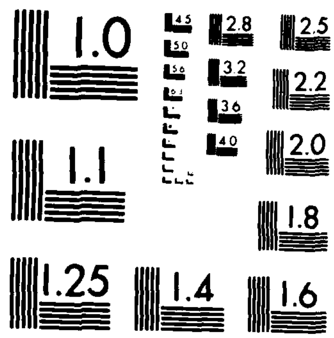


Table III. Sensitivity Analysis Results

Crit. #	Weight	Rel. Total	EK Total	Canon. Total
Start	30	70	70	55
2a	5	71	73	57
2b	3	69	67	53
6b	4	67	68	54
2a,6b	9	68	71	56
2b,6b	7	67	65	52
Minus 4	27	67	61	46
2a	5	68	64	48
2b	3	66	58	44
6b	4	64	59	45
2a,6b	9	65	62	47
2b,6b	7	64	56	43

An examination of the above results indicates that the original criteria set (criteria #4 included) was very sensitive to changes in the weight factors. The second criteria set (criteria #4 removed) is much less sensitive to these changes and shows the relational model to be the better choice as the universal model (according to the given criteria and weights).

Final Design Decisions

In the course of this thesis, many issues have been raised about the environment, users, and so forth. The way in which these issues were to be resolved was to a great ex-

tent dependent on the model chosen for the UDM. Therefore, having made that decision, the following final design decisions are presented.

Design Decision 1: Local users will be required to utilize the UDBMS for queries which involve global data. Local users will still be able to utilize their local model/DML for "local only" queries.

Design Decision 2: The data contained within the global system will be presented to the user in a relational format.

Design Decision 3: The UDB, when evaluating any query in the local DML, will notify the user if and when there is additional data in the UDBMS for that local query.

Design Decision 4: Procedural operations will not be supported in the UDBMS. Procedural operations will be mapped from the UDML to those LDBMS supporting procedural operations but the user may not write UDML commands which "navigate" through the UDB.

Design Decision 5: The UDBMS will be a relationally based language but not necessarily any presently designed system.

Design Decision 6: The UDML will support embedded and interactive capabilities which are syntactically similiar.

Design Decision 7: Direct Reference will be supported.

Design Decision 8: Null values will be supported provided that those values are not primary key values.

Design Decision 9: The UDML will have separate constructs for selection and action specification.

Design Decision 10: The UDML will support selection nesting.

Design Decision 11: The UDML will support a form of record-at-a-time capability. The records will be acquired a set-at-a-time but may be analyzed

individually in the action specification portion of a query.

The Universal Data Definition Language

The relational DML specified for the UDB is based on System R (see Date (3)). An example data base is defined below:

```
( Create Data Base MEDICAL DATA BASE (

Domain char-15 Character (15)
Domain char-20 Character (20)
Domain code-type Integer (NONULL)
Domain #type Integer (NONULL)
Domain small-int Small Integer
Domain var-string Character Var

Create HOSPITAL (
     Unique Hospital code:  code-type,
     name:  char-15,
     address:  char-20,
     phone#:  #type,
     # of beds (Small Integer),
     Keys are Hospital code )

Create LAB (
     Unique Lab#:  #type,
     Hospital code:  code-type,
     name:  char-20,
     address:  char-20,
     phone#:  #type,
     Keys are Lab# )

Create PATIENT-DOCTOR
     Alternate names are ATTENDING DOCTOR
       (*  Treat as normal relation *)
     Alternate names are PAT-ATTD
       (*  List of patients a given doctor is seeing *)
     Alternate names are PATIENTS ATTENDED
       (*  List of patients a given doctor is seeing *)
     Alternate names are ATT-DOC
       (*  List of doctors attending given patient *)
     Alternate names are DOCTORS ATTENDING
       (*  List of doctors attending given patient *)
     Unique Doctor#:  #type,
     Unique Registration#:  #type,
     Keys are All )
) )
```

```
( Expand Data Base MEDICAL DATA BASE
(  User Function LABS-SERVING-A-HOSPITAL
   Input Arguments are (hosp-name:  code-type)
   Output Arguments are (lab-name:  #type)
   Retrieve
   From  LAB known by L,
         HOSPITAL-LAB known by HL,
         HOSPITAL known by H,
     ( Where hosp-name = H.name and
         H.Hospital code = HL.Hospital code and
         L.Lab# = HL.Lab#
       ( Return (L.name Ordered by Ascending L.name))
  END FUNCTION ) )
```

The Universal Data Manipulation Language

The relational DML specified for the UDB is based on a language called QUEST developed by Housel (5). Sample queries are presented below:

1. List all hospitals and their addresses.

```
( Retrieve
  From HOSPITAL known by H,
  ( Return ( H.name, H.address ) ) )
```

2. List all doctors serving in hospitals with over 250 beds.

```
( Retrieve
  From HOSPITAL known by H,
       DOCTOR known by D,
    ( Where H.# of beds > 250 and
      H.Hospital code = D.Hospital code
      ( Return (D.name) ) ) )
```

3. List all doctors serving in hospitals in Tollersville.

```
( Retrieve
  From HOSPITAL known by H,
       DOCTOR known by D,
       HOSPITAL LOCATION known by HL,
       ( Where H.Hospital code = D.Hospital code
         and HL.code = "TLV"
         ( Return (D.name) ) ) )
```

4. List all doctors who attend over 15 patients and list those patients under each doctor's name.

```
( Retrieve
  From DOCTORS known by D,
       DOCTOR-PATIENT known by DP,
  ( Where COUNT (D.Doctor# = DP.Doctor# ) > 15
    Return (D.name, 2RLF,
    ( Retrieve
      From  PATIENT known by P,
      ( Where D.Doctor# = DP.Doctor# and
        P.Registration# = DP.Registration#
        ( Return (P.name ordered by Ascending P.name,
          2RLF) ) ) ) ) )
```

5. Add a 5% pay raise to all staff employees earning over $16,000 and a 10% raise to those earning under $16,000. E shift employees earn an additional 2% pay raise.

```
( Update
  From  STAFF known by S,
   ( Case
       S.salary > 16000:
         ( Case
            S.shift = 'E':  S.salary := S.salary * 1.07,
            Otherwise:  S.salary := S.salary * 1.05 )
       Otherwise:
         ( Case
            S.shift = 'E':  S.salary := S.salary * 1.12,
            Otherwise:  S.salary := S.salary * 1.1 )
    ) )
```

Universal Data Definition Language Mappings

Many different important issues were raised when the UDDL mappings were examined. The conclusions of these issues are summarized below:

Redundant data: The UDB will allow for partial redundancy in the underlying data bases.

Universal Query Processing: The UDB will possess actual processing capability at the universal level. The UDBAC computer will provide this capability when needed. When a query requires that various subqueries be brought together, this will be done by the UDBAC computer (a data base computer) or a relational computer on the network.

System R Constraints: (1) All system R systems will be required to specify which attributes within a relation are the primary or composite keys. (2) The System R systems are as described by Date (3).

IMS Constraints: (1) All IMS systems will be required to specify which attributes within a segment uniquely identify the information within that segment. (2) All IMS systems will not allow duplicate keys. (3) The IMS systems are as described by Date (3).

DBTG Constraints: (1) All DBTG systems will be required to specify which attributes within a record uniquely identify the information within that record. (2) Fixed retention is a user-policed retention class. (3) Automatic insertion is not supported. (4) Duplicate keys are not allowed. (5) The DBTG system is as per described by Date (3).

Normal Forms: Only first normal form is guaranteed by the UDB. This is because of underlying constraints in the IMS and DBTG data bases.

DBTG to UDDL Mapping Algorithm: (1) Records are converted directly to a relational format. (2) Structural sets are also converted directly to a relational format. (3) Sets which have an optional retention class are converted into a relational format. (4) All remaining sets are used to indicate where to add the primary key of the owner to the member record, now relation. All redundant attributes and relations are removed.

IMS to UDDL Mapping Algorithm: (1) All segments are converted to a relational format. (2) All children add the primary key of their parent to their relation. (3) Redundant attributes and relations are removed. (4) Added attributes are removed if when paired with another attribute in more than one relation.

System R to UDDL Mapping Algorithm: Merely a snytax translation.

Conclusion

This section concludes this paper by commenting on the UDM and examining what was accomplished in this thesis and what was not.

System R Constraints: (1) All system R systems will be required to specify which attributes within a relation are the primary or composite keys. (2) The System R systems are as described by Date (3).

IMS Constraints: (1) All IMS systems will be required to specify which attributes within a segment uniquely identify the information within that segment. (2) All IMS systems will not allow duplicate keys. (3) The IMS systems are as described by Date (3).

DBTG Constraints: (1) All DBTG systems will be required to specify which attributes within a record uniquely identify the information within that record. (2) Fixed retention is a user-policed retention class. (3) Automatic insertion is not supported. (4) Duplicate keys are not allowed. (5) The DBTG system is as per described by Date (3).

Normal Forms: Only first normal form is guaranteed by the UDB. This is because of underlying constraints in the IMS and DBTG data bases.

DBTG to UDDL Mapping Algorithm: (1) Records are converted directly to a relational format. (2) Structural sets are also converted directly to a relational format. (3) Sets which have an optional retention class are converted into a relational format. (4) All remaining sets are used to indicate where to add the primary key of the owner to the member record, now relation. All redundant attributes and relations are removed.

IMS to UDDL Mapping Algorithm: (1) All segments are converted to a relational format. (2) All children add the primary key of their parent to their relation. (3) Redundant attributes and relations are removed. (4) Added attributes are removed if when paired with another attribute in more than one relation.

System R to UDDL Mapping Algorithm: Merely a snytax translation.

Conclusion

This section concludes this paper by commenting on the UDM and examining what was accomplished in this thesis and what was not.

An Augmented Relational Model

As the UDB system was analyzed and developed through this thesis, it became obvious that the relational model, as it stood, would require some modifications to fully satisfy the UDB requirements. The reason for these modifications come from attempting to map nonrelational structures and operations into a relational model. It is thought that it will require even further modifications after the DML requirements are fully analyzed. The present modifications, relatively minor, do not change the basic nature of the relational model, they merely augment it. It is suggested that the additional integrity constraints suggested by Date (3) be fully supported. The following are characteristics of the the augmented relational model used:

1. Guarantees only 1NF.
2. Alternate names construct to handle DBTG structural sets.
3. Unique associations clause to alert users to a DBTG fixed retention set.

The following additional characteristics are anticipated:

1. Support of Domain Integrity.
2. Support of Immediate Record State Constraint.
3. Support of Immediate Record Transition Constraint.
4. Constraint or construct to support DBTG Automatic insertion.
5. Constraint or construct to support the Fixed Retention class in the DBTG DBMS.

Accomplishments. Although this thesis has not implemented any part of the UDB nor has it really fully investigated all of the issues raised, it does provide a good starting point for further investigation. The following list indicates what was accomplished in this thesis:

1. Literature search of current research into the area of a UDM and/or UDB.
2. An analysis of the requirements for a UDB.
3. The selection of a UDM.
4. An examination of the UDDL mapping issues.
5. Syntax specification for a UDML and UDDL.

Universal Data Model Deficiencies. Although the Relational model was chosen as the best model, of those examined, it is obvious that the UDM, and UDB, as presented in this thesis have several deficiencies or otherwise undesired qualities. It is hoped that these undesired qualities will be eliminated by the time the UDB is actually implemented. The following list summarizes those deficiencies:

1. The UDB is restricted to dealing with one particular implementation of each of the three different models.
2. Each of those three particular implementations (IMS, DBTG, and System R) have "unnatural" constraints imposed upon them by the UDB.
3. The users of the UDB are forced to work in the UDML.
4. Using the UDB may require modifications to the underlying data bases.
5. DML mappings have not been examined to insure that they can be completely supported.

6. The DDL mappings examined have not been fully tested to insure that they are complete and accurate.

7. The relational model used in this thesis could, perhaps, be augmented further to perform better. This reevaluation should be done after the DML requirements have been closely examined.

Concluding Comments. The concept of a UDB has great potential to radically increase the power and usefulness of DBMSs. This thesis has taken the first step towards the complete development of the UDB and has provided the necessary foundation for its continued development.

Bibliography

1. Cardenas, A. F. and Pirahesh, M. H. "The E-R Model in a Heterogeneous Data Base Management System Network Architecture," Entity-Relationship Approach to Systems Analysis and Design, edited by P. P. Chen. Amersterdam: North-Holland Publishing Company, 1980.

2. Chen, Peter P. "The Entity-Relationship Model--Toward a Unified View of Data," ACM Transactions on Database Systems, 1: 9-36 (March 1976).

3. Date, C. J. An Introduction to Database Systems, third edition. Addison-Wesley Publishing Company, Inc., Reading, Massachusetts, 1982.

4. Date, C. J. "An Introduction to the Unified Database Language (UDL)," Sixth International Conference on Very Large Data Bases, 15-32 (1980).

5. Housel, B. C. "QUEST: A High-Level Query Language for Network, Hierarchical, and Relational Databases," Improving Database Usability and Responsiveness, edited by Peter Scheuermann, New York: Academic Press, 1982.

6. Huang, Kuan-Tsae, and Davenport, Wilbur B. "Query Processing in Distributed Heterogeneous Databases," LIDS-P-121, Department of Electrical Engineering and Computer Science and the Laboratroy for Information and Decision Systems, M.I.T., Cambridge, Mass, 02139 (December 1981) (AD-115981).

7. Larson, James A. "Bridging the Gap Between Network and Relational Database Management Systems," IEEE Computer, 16: 82-92 (September 1983).

8. Lillie, Major Charles W. "The Coalescing of Network Structured Database Performance with Relational Algebra Productivity," Dissertation. School of Arts and Sciences, University of Soutwestern Louisiana. (Not yet published).

9. Lochovsky, Frederick H., and Tsichritzis. Data Models. Prentice-Hall, Inc., Englewood Cliff, New Jersey, 1982.

10. Martin, James. Principles of Data-Base Management. Prentice-Hall, Inc., Englewood Cliff, New Jersey, 1976.

11. Pressman, Roger S. Software Engineering: A Practioner's Approach. McGraw-Hill Book Company, New York, New York, 1982.

12. Ullman, Jeffrey D. Principles of Database Systems, second editon. Computer Science Press, Rockville, Maryland, 1982.

VITA

Second Lieutenant Anthony J. Jones was born on 13 June 1961 in Muscatine, Iowa. He graduated from Kaiserslautern American High School, West Germany, in 1979 and attended Indiana University from which he recieved the degree of Bachelor of Arts in Computer Science in May 1983. Upon graduation, he received a commission in the USAF through the ROTC program. He entered the School of Engineering, Air Force Institute of Technology, in May 1983.

Permanent address: 1487 Shorewood Place
Lakeland, Florida 33803

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION: Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release; IAW AFR 190-17. LYNN E. WOLAVER, Dean for Research and Professional Development, Air Force Institute of Technology (AIC), Wright-Patterson AFB
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S): AFIT/GCS/ENG/84D-11	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION: School of Engineering; 6b. OFFICE SYMBOL (If applicable): AFIT/ENG	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State and ZIP Code): Air Force Institute of Technology, Wright-Patterson AFB, OH 45433	7b. ADDRESS (City, State and ZIP Code)
8a. NAME OF FUNDING/SPONSORING ORGANIZATION: Rome Air Development Center; 8b. OFFICE SYMBOL (If applicable): RADC/COTD	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State and ZIP Code): HQ Rome Air Development Center, Griffiss AFB, NY 13441	10. SOURCE OF FUNDING NOS. — PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT NO.
11. TITLE (Include Security Classification): See box 19.	
12. PERSONAL AUTHOR(S): Anthony J. Jones, B.A., 2LT, USAF	
13a. TYPE OF REPORT: M.S. Thesis; 13b. TIME COVERED: FROM _____ TO _____; 14. DATE OF REPORT (Yr., Mo., Day): 84/12/14; 15. PAGE COUNT: 299	
16. SUPPLEMENTARY NOTATION	

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB. GR.	Data Base, Relational Data Base, Data Model, Heterogeneous Environment, Distributed Environment.
09	02		

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Title: Analysis and Specification of a Universal Data Model for Distributed Data Base Systems.

Thesis Chairman: Dr. Thomas C. Hartrum

Approved for public release; IAW AFR 190-17.
Lynn E. Wolaver 21 Feb 85
Dean for Research and Professional Development

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT: UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS ☐	21 ABSTRACT SECURITY CLASSIFICATION: Unclassified	
22a. NAME OF RESPONSIBLE INDIVIDUAL: Dr. Thomas C. Hartrum	22b TELEPHONE NUMBER (Include Area Code): 513-255-3576	22c OFFICE SYMBOL: AFIT/ENG

DD FORM 1473, 83 APR — EDITION OF 1 JAN 73 IS OBSOLETE.

19. A Universal Data Model (UDM) was developed for distributed Data Base Management Systems (DBMS). The primary goal was to allow for the effective communication between heterogeneous, distributed DBMSs. A system requirements analysis was first performed for a Universal Data Base (UDB). Three models were selected and investigated as candidates for the UDM: the Canonical, Entity-Relationship, and Relational. Due to the complexities of the UDB, the user was restricted to writing universal queries in a Universal Data Manipulation Language (UDML) and was restricted to only one version of each of the three prominent data models in use: IMS (heirarchical), DBTG (network), and System R (relational). Criteria were established and the relational model, augemented, chosen as the UDM. Data model mapping issues were examined and included discussions on distributed information, redundant data, the support of Third Normal form, and target model specific issues. Algorithms were developed to show the mappings between the target models and the UDM. The integration of these mappings were also addressed. The syntax of a universal data definition language and data manipulation language were described.

END

FILMED

5-85

DTIC